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

Manindra Agrawal   Anil Seth (Eds.)

FST TCS 2002: Foundations of Software Technology and Theoretical Computer Science

22nd Conference
Kanpur, India, December 12-14, 2002
Proceedings

Springer

Series Editors

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

Volume Editors

Manindra Agrawal
Anil Seth
Indian Institute of Technology
Department of Computer Science and Engineering
Kanpur 208016, India
E-mail: {manindra,seth}@iitk.ac.in

Cataloging-in-Publication Data applied for

A catalog record for this book is available from the Library of Congress

Bibliographic information published by Die Deutsche Bibliothek
Die Deutsche Bibliothek lists this publication in the Deutsche Nationalbibliographie; detailed bibliographic data is available in the Internet at <http://dnb.ddb.de>.

CR Subject Classification (1998): F.3, D.3, F.4, F.2, F.1, G.2

ISSN 0302-9743
ISBN 3-540-00225-1 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author, data conversion by Olgun Computergraphik
Printed on acid-free paper SPIN: 10871762 06/3142 5 4 3 2 1 0

Preface

This volume consists of the proceedings of the 22nd International Conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2002), organized under the auspices of the Indian Association for Research in Computing Science (IARCS). The conference was held at the Indian Institute of Technology, Kanpur during December 12–14, 2002.

The conference attracted 108 submissions (of which two were withdrawn). Of these, a total of 26 papers were selected for presentation in the conference. As in the last year, the PC meeting was held electronically (stretching over nearly three weeks in August 2002) and was a great success.

In addition to the contributed papers, we had five invited speakers this year: Hendrik Lenstra, Jr., Harry Mairson, Dale Miller, Chih-Hao Luke Ong, and Margus Veanes. We thank them for accepting our invitation and for providing abstracts (or even full papers) for the proceedings.

Two workshops were organized in conjunction with the conference – both in Kanpur. A workshop on Parameterized Complexity was held during December 10–11, organized by Mike Fellows and Venkatesh Raman. The second workshop actually consisted of three miniworkshops: on Coding Theory by Madhu Sudan; on Finite Field Algorithms by Hendrik Lenstra, Jr.; and on Sieve Theory by R. Balasubramanian.

We wish to thank all the reviewers and PC members who contributed greatly to making the conference a success. We also wish to thank the team at Springer-Verlag for their help in preparing the proceedings.

December 2002

Manindra Agrawal
Anil Seth

Organization

FSTTCS 2002 was organized by the department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, in cooperation with IARCS.

Program Committee

Organizing Committee

Referees

Bharat Adsul
Gagan Aggarwal
Roberto Amadio
Giambattista Amati
Torben Amtoft
Richard Anderson
Sergio Antoy
V. Arvind
Nick Benton
Josh Berdine
Stefan Berghofer
Therese Biedl
Somenath Biswas
Genest Blaise
Bruno Blanchet
Bernard Boigelot
Julian Bradfield
Cristiano Calcagno
Didier Caucal
L. Sunil Chandran
Chandra Chekuri
Benny Chor
Peter Clote
Amitava Datta
Anuj Dawar
Erik Demaine
Kedar Dhamdhere
Rod Downey
Aldrin John D'Souza
Javier Esparza
Kousha Etessami
Henning Fernau
Melvin Fitting
Arnaud Fleury
P. Fradet
Martin Fraumlnzle
Thom Fruehwirth
Vijay K. Garg
William Gasarch
Simon Gay
Thomas Genet
Konstantinos Georgatos
Aristides Gionis
Valentin Goranko
Andy Gordon
Daniele Gorla
R. Govindarajan
Sudipto Guha
Joshua D. Guttman
Hari Hampapuram
Peter G. Hancock
Rene Rydhof Hansen
T. Hardin
Harald Hempel
Kohei Honda
Hans Hüttel
Sanjay Jain
David Janin
Leroux Jerome
Felix Joachimski
R.K. Joshi
Steffen Jost
Sara Kalvala
V. Kamakoti
Hartmut Klauck
Adam Klivans
Neal Koblitz
Jochen Konemann
Maciej Koutny
Narasimha Mandyam Krishnakumar
Ramesh Krishnamurti
S. Krishnan
Armin Kühnemann
Sandeep Kulkarni
D. Phaneendra Kumar
K. Narayan Kumar
Viktor Kuncak
Piyush P. Kurur
Orna Kupferman
Dietrich Kuske
Martin Kutrib
Martin Lange
Luciano Lavagno
Ranko Lazic
Ming Li
Kamal Lodaya
Hans-Wolfgang Loidl
Gavin Lowe
Christoph Lüth
Anil Maheshwari
Monika Maidl
Harry Mairson
Rupak Majumdar
Russell Martin
Fabio Martinelli
Jiri Matousek
Francois Maurel
B. Meenakshi
M. Merro
Stephan Merz
Ron van der Meyden
Peter Bro Miltersen
S. Mishra
Kundan Misra
Swarup Kumar Mohalik
Mark Moir
Christophe Morvan
Pablo Moscato
Jose M. Sempere
Madhavan Mukund
Andrzej Murawski
H. Nagamochi
N.S. Narayanaswamy
Ashwin Nayak
Ilan Newman
Peter Niebert
Mogens Nielsen
Liadan O'Callaghan
Mitsunori Ogihara
Mitsuhiro Okada
Luke Ong
Friedrich Otto
Paritosh K. Pandya
Rina Panigrahy
Kapil Hari Paranjape
Ojas Parekh
Rohit Parikh
Dirk Pattinson
Jean-Guy Penaud
Iain Phillips
David Plaisted

Randy Pollack
Jaikumar Radhakrishnan
Balaji Raghavachari
Krithi Ramamritham
Rajeev Raman
S. Ramesh
Abhiram G. Ranade
Julian Rathke
Uday Reddy
Klaus Reinhardt
Eike Ritter
Stefan Roock
Peter Rossmanith
Frank Ruskey
Ivano Salvo
Prahladavaradan Sampath
Amitabha Sanyal
Pranab Sen
Sanjit Seshia
Paula Severi
Priti Shankar
Dmitry Shkatov
Amitabh Sinha
Rani Siromoney
Milind Sohoni
Paul Spirakis
Aravind Srinivasan
Alin Stefanescu
Ulrike Stege
S.P. Suresh
Scott D. Stoller
Martin Strecker
C.R. Subramanian
P.R. Subramanya
K.V. Subrahmanyam
S. Sudarshan
Peter J. Taillon
Olivier Tardieu
Kazushige Terui
P.S. Thiagarajan
Thomas Thierauf
Wolfgang Thomas
Tayssir Touili
Ralf Treinen
Vincent Vanackere
Daniele Varacca
Kasturi Varadarajan
Moshe Vardi
Umesh Vazirani
Santosh Vempala
Sundar Vishwanathan
Vinodchandran N. Variyam
Maria Grazia Vigliotti
Erik de Vink
Fer-Jan de Vries
Todd Wareham
Pascal Weil
Thomas Wilke
An Zhu

Sponsoring Institution

Indian Institute of Technology, Kanpur

Table of Contents

Primality Testing with Gaussian Periods

H.W. Lenstra, Jr.

University of California at Berkeley
hwl@math.berkeley.edu
and
Universiteit Leiden
hwl@math.leidenuniv.nl

Abstract. It was recognized in the mid-eighties, that several then current primality tests could be formulated in the language of Galois theory for rings. This made it possible to combine those tests for practical purposes. It turns out that the new polynomial time primality test due to Agrawal, Kayal, and Saxena can also be formulated in the Galois theory language. Whether the new formulation will allow the test to be combined with the older tests remains to be seen. It does lead to a primality test with a significantly improved guaranteed run time exponent. In this test, one makes use of Gaussian periods instead of roots of unity. The lecture represents joint work with Carl Pomerance (Bell Labs).

From Hilbert Spaces to Dilbert Spaces: Context Semantics Made Simple

Harry G. Mairson[⋆]

Computer Science Department
Brandeis University
Waltham, Massachusetts 02454
mairson@cs.brandeis.edu

Abstract. We give a first-principles description of the *context semantics* of Gonthier, Abadi, and Lévy, a computer-science analogue of Girard's *geometry of interaction.* We explain how this denotational semantics models λ-calculus, and more generally multiplicative-exponential linear logic (MELL), by explaining the call-by-name (CBN) coding of the λ-calculus, and proving the correctness of *readback*, where the normal form of a λ-term is recovered from its semantics. This analysis yields the correctness of Lamping's optimal reduction algorithm. We relate the context semantics to linear logic types and to ideas from *game semantics*, used to prove full abstraction theorems for PCF and other λ-calculus variants.

1 Introduction

In foundational research some two decades ago, Jean-Jacques Lévy attempted to characterize formally what an *optimally efficient* reduction strategy for the λ-calculus would look like, even if the technology for its implementation was at the time lacking [13]. For the language designer, the λ-calculus is an important, canonical abstraction of the essential features required in a programming language, just as the classical physicist (or freshman physics student) views the world via a model of massless beams and frictionless pulleys, and as machine architects have the Turing machines as their essential, ideal archetype.

Lévy wanted to build a λ-calculus interpreter that was *correct* and *optimal.* Every interpreter specifies an *evaluation strategy*, determining what subexpression is evaluated next. A *correct* interpreter never chooses a subexpression whose evaluation is *divergent* unless there is no other choice, so that it produces a *normal form* (answer) if there is one. An *optimal* evaluator shares *redexes* (procedure calls) in a technically maximal sense: the problem here is that evaluation can easily *duplicate* redexes, for example in $(\lambda x.xx)((\lambda w.w)y)$.

John Lamping [12] found the algorithm that Lévy specified. Then Gonthier, Abadi, and Lévy [9,10] made a lovely discovery: they gave a denotational semantics to Lamping's algorithm, called *context semantics*, and showed that it

[⋆] Supported by NSF Grants CCR-0228951 and EIA-9806718, and also by the Tyson Foundation.

was equivalent to Jean-Yves Girard's *geometry of interaction* (GoI) [8]. Girard's GoI is an abstract mathematical notion, employing a lot of higher mathematics that theoretical computer scientists are not accustomed to using: Hilbert spaces, C^*-algebras, and more. In contrast, context semantics is built from familiar, ordinary data structures: stacks and trees of tokens. The vectors of a Hilbert space are just a data structure, and the linear operators Girard designed were just ways of hacking the data structure. Context semantics provides a precise flow analysis, where static analysis is effected by moving *contexts* (a data structure) through a fixed program graph, so that questions like "can call site α ever call function f?" become straightforward to answer. Girard's matrix *execution formula* modulo a *path algebra* is then just transitive closure of a directed graph (just like in an undergraduate automata theory class), where the algebra rules out certain paths. The GoI formalism is not unlike the use of generating functions to analyze combinatorial problems. Computer scientists should find context semantics to be a more compelling formalism, because there is no math, and the resemblance to static program analysis is so strong.

The main purpose of this paper is to give a simple explanation of context semantics—it's the paper I wish I could have *read* six or eight years ago. I particularly want to give a first-principles proof of the algorithmic correctness of optimal reduction, via a naive explanation of *readback*: the process whereby the normal form of a λ-term is recovered from its context semantics. *A key goal is to explain how GoI, and context semantics, and games, and full abstraction, are founded on technical, detailed ways of talking about Böhm trees (term syntax);* like Molière's Monsieur Jourdain, we should know prose when we are speaking it. This observation is not meant to be dismissive—I want to explain how context semantics details the mechanics of reduction and the extraction of answers, via static analysis. I want to explain how linear types give a refined view of contexts. Most important, I want the discussion to be as informal and intuitive as possible—like a conversation—with incomplete proofs and basic intuitions.

For those who from semantics recoil, recall: this is algorithm analysis, establishing the correctness of an incremental interpreter. Related complexity analysis, both of Lamping's algorithm and the problem, can be found in [6,4].

2 Linear λ-Terms

The best place to start—representing the essence of the full problem, but in miniature—is with *linear* λ-terms: every variable must occur exactly once. Church numerals $\lambda s.\lambda z.s^n z$ are ruled out (s may occur more than once or not at all), also Boolean truth values (recall $\mathrm{T} \equiv \lambda t.\lambda f.t$, $\mathrm{F} \equiv \lambda t.\lambda f.f$ discard an argument)[1]. We now code terms as graphs—or *proofnets* in the terminology of linear logic. Represent application (@) and λ-abstraction by ternary nodes: the @-node

[1] Yet the language still has expressive power: for example, we could define True and False as $\lambda t.\lambda f.\langle t, f\rangle$ and $\lambda t.\lambda f.\langle f, t\rangle$, where $\langle x, y\rangle \equiv \lambda z.zxy$. Then let $\mathsf{Not} \equiv \lambda p.\lambda t.\lambda f.pft$ and $\mathsf{Or} \equiv \lambda p.\lambda q.\lambda t.\lambda f.p\,\mathsf{True}\,q(\lambda u.\lambda v.vIIIu)$. Add a fanout gate, and we can simulate circuits; thus evaluation is complete for PTIME.

has *continuation* and *argument ports*, which are called *auxiliary*, and a *function* port, which is called *principal*. A λ-node has auxiliary *body* and *parameter* ports, and a principal *root* port. Since a bound variable occurs exactly once, the wire coding its *occurrence* is connected to the λ-node coding its *binder*. A β-reduction happens when an edge connects an @- and λ-node by their principal ports: the graph is rewritten so that these nodes annihilate, and we fuse the wires previously connected with the @-continuation and the λ-body, and the λ-parameter and the @-argument (see Figure 1).

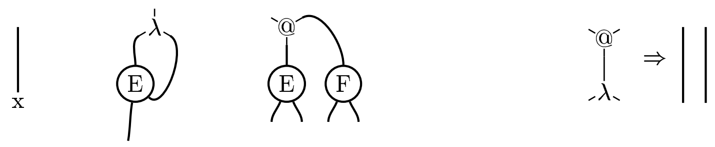


Fig. 1. Coding and reducing linear λ-terms as graphs.

Proposition 1. *Let G_Λ denote the graph coding of λ-term Λ. Then $E \to E'$ iff G_E reduces to $G_{E'}$ by the above annihilation and rewiring.*

Now interpret each graph node as a transformer on primitive *contexts* $c \in \Sigma^*$ (where $\Sigma = \{\circ, \bullet\}$, called *tokens*) which travel on graph wires. Entering an @-continuation or λ-body port transforms c to $\circ c$; entering an @-argument or λ-parameter port transforms c to $\bullet c$, and the context emerges at the respective principal port. Dually, entering an @-node (λ-node) at the principal @-function (λ-root) port with context τc, the token τ is popped, and the context c emerges at the @-continuation (λ-body) port if $\tau = \circ$, and at the @-argument (λ-parameter) port if $\tau = \bullet$. Notice that as context transformers on Σ^*, @- and λ-nodes are implemented by exactly the same hardware, which pushes and pops tokens, with routing to appropriate auxiliary ports in the latter case; we call this hardware a *fan node.*

The *ports* of a λ-term E (and its graph coding) are the dangling wire ends that represent the *root* and each free variable v of E; call these ports ρ and π_v. E has a *context semantics*—the relation given by paths between graph ports, where a path is defined by a *context* that is itself transformed by nodes along the path: $\mathcal{C}_E = \{(\langle c, \pi\rangle, \langle c', \pi'\rangle) \mid \text{context } c \text{ enters at port } \pi \text{ and exits as } c' \text{ at port } \pi\}$.

Proposition 2. *$\mathcal{C}_E$ is symmetric, and is preserved by β-reduction.*

Proof. The nodes forming a β-redex transform context c by pushing and then popping either $\circ$ (between an @-continuation and λ-body) or $\bullet$ (between an @-argument and λ-parameter), leaving c unchanged. The resulting paths are described by the wire fusings above[2].

[2] For this reason, the path algebra of Girard's GoI include equations like $p^*p + q^*q = 1$ (pushing and then popping $\circ$ [or $\bullet$] is the identity transformation), or $q^*p = 0$ (you cannot push $\circ$ and then pop $\bullet$).

Linear λ-calculus is strongly normalizing, as reduction makes the λ-term (graph) get smaller. Since reduction preserves context semantics, the context semantics of a λ-term identifies its normal form. We now consider the process of *readback*: how can we reconstruct the normal form of E from $\mathcal{C}_E$?

Readback for linear λ-calculus. Let $N \equiv \lambda x_0. \cdots \lambda x_p.vN_0 \cdots N_q$ be the normal form of E. Let $c_0 = \circ^+$ be some arbitrarily large string (stack) of $\circ$ tokens, enough so that, informally, as c_0 (or later, some variant of it) traces a path through the graph, the context is never empty when entering the principal port of a node, "stopping" the path. (We also write $\circ^k$ for the word consisting of k $\circ$ tokens.) If we add or subtract some small number of $\circ$ tokens to c_0, we don't care—we still write it as $\circ^+$. Insert c_0 at the root ρ: if $(\langle c_0, \rho\rangle), \langle c, \pi\rangle) \in \mathcal{C}_E$, what does $\langle c, \pi\rangle$ tell us about the normal form of E?

Proposition 3. *Context c identifies the* head variable *of N: $v = x_i$ iff $\langle c, \pi\rangle = \langle \circ^i \bullet \circ^+, \rho\rangle$, and v is a free variable if $\langle c, \pi\rangle = \langle \circ^+, \pi_v\rangle$.*

If $v = x_i$, then the *address* α of the head variable is $\circ^i\bullet$, otherwise the address is ϵ. Starting at port π above, $\alpha \circ^j \bullet$ defines a path ending at the root of subterm N_j. Recursing on this construction, we examine $\mathcal{C}_E$ for $(\langle \alpha \circ^j \bullet \circ^+, \pi\rangle, \langle c', \pi'\rangle)$; as in Proposition 3, c' identifies the head variable of $N_j \equiv \lambda y_0. \cdots \lambda y_r.wF_0 \cdots F_s$: observe that $\langle c', \pi'\rangle$ is $\langle \circ^+, \pi_w\rangle$ if w is free, $\langle \circ^{i'} \bullet \circ^+, \rho\rangle$ if $w \equiv x_{i'}$, and finally $\langle \alpha \circ^j \bullet \circ^{j'} \bullet \circ^+, \pi\rangle$ if $w \equiv y_{j'}$. Thus contexts can be *decoded* to recover the head variables and subterms of all subexpressions of N.

Readback can be thought of as a *game* between an Opponent (Op) who wants to discover the normal form of a λ-term known by the Player (Pl). The *initial move* $\langle c_0, \rho\rangle$ codes the Op-question, "what is the head variable of the term?" The Pl-answer $\langle c, \pi\rangle$ is then *transformed* (by splicing $\circ^j\bullet$ into the context just before the $\circ^+$) into the next Op-question "what is head variable of the jth argument of the head variable?" and so on[3]. The Op-moves identify the addresses of subterms, the Pl-moves those of their head variables. A "winning strategy" for the Player means she can always respond to an Op-move; each λ-term codes a winning strategy for the Player—it gives her a λ-term to talk about. Games in the style of [1,11] are essentially this guessing of normal forms. We use this *argot* to describe the input and output of contexts at graph ports.

How does the readback algorithm terminate? In linear λ-calculus, η-expansion preserves context semantics: the graph of $\lambda x.Ex$ pops and then pushes a token at its root. Thus there is no termination if the semantics is infinite: we just generate ever-larger pieces of the *Böhm tree*, a computation which quickly becomes boring. However, we can (nonconstructively) consider a *minimal* subset $\mathcal{C}^0 \subseteq \mathcal{C}$ where $(\langle c, \pi\rangle, \langle c', \pi'\rangle) \in \mathcal{C}^0$ iff for all $\sigma \in \Sigma^*$, $(\langle c\sigma, \pi\rangle, \langle c'\sigma, \pi'\rangle) \in \mathcal{C}$. Then readback terminates when all elements of $\mathcal{C}^0$ have been used.

What about terms not in normal form? In this case, readback constructs the normal form from contexts at the *ports*—but what do contexts mean as they

[3] The generation of Op-moves from Pl-answers is called *shunting* in [9], evoking a train at a head variable shunting off a "track" of applications to one of its arguments.

travel through the interior of the graph? A "call" to a subgraph S, intuitively, requests the subterm of the graph that is bound to the head variable of the normal form of S. Consider a term $(\lambda x.E)F$; focus on the contexts that travel on the wire *down* from the @-node to the function $\lambda x.E$, and on the same wire *up* from the function to the @-node. First, a "call" is made *down* to the function; if its head variable h_0 is the parameter, a context travels *up* that is the address $\bullet$ of the variable. Then the call is made to the argument F; if its head variable h_1, is bound, its address is routed back to h_0 to find the subterm (an argument of h_0) that is bound to h_1. In turn, the address of the head variable h_2 of that subterm may be routed back to h_1 in search of its bound subterm. And so on: head variable addresses in $\lambda x.E$ are used to find subterms in F and vice versa—like Player and Opponent, the two terms play against each other.

Finally, we should mention the connection to linear logic [7]. The graphs described are *proofnets* for *multiplicative linear logic*, which has logical connectives $\otimes$ (conjunction) and $\parr$ (disjunction). The $\otimes$ (alias @) *constructs a pair* from a continuation and an argument; the $\parr$ (alias λ) *unpairs* them for the body and parameter of the function. But if the nodes are logical connectives, what are the formulas? We discuss this briefly in Section 4.

3 Sharing

Now let's generalize linear λ-calculus, and allow multiple references to a bound variables. Three problems arise: What do we do when a bound variable is *not* referenced? How do we *combine* references? And *what* is being shared? To address these implementation issues, the graph technology, and its context semantics, get more complicated. First, some brief answers: In the case of no reference, we attach a unary *plug node* to the parameter: for example, in $\lambda x.\lambda y.x$, a plug is attached to the end of the wire coding y. Multiple references are combined with a fan node—the same kind of hardware used for @ and λ-nodes; we call it a *sharing node*. Expressions being shared by a bound variable are encapsulated in a *box*. To be used, a box needs to be *opened*; we introduce a binary *croissant node* that opens (erases) boxes. Boxed expressions may contain references to other boxed values; we thus introduce a binary *bracket node* to *absorb* one box into another, and a means to *fuse* one box with another. These ideas are now presented in the context of the call-by-name version of the λ-calculus.

Call-by-name coding and graph reduction. Figure 2 illustrates the coding; we write G_E for the coding of λ-term E. Observe that each free variable is represented by a single port. A *variable* v is coded as a croissant; when reduction attaches the wire for v to the boxed expression it is bound to, the croissant opens the box. An *abstraction* $\lambda x.E$ modifies G_E, using a λ-node to connect its root with the port of free variable x. An *application* EF is coded by coding E and F connected by an @-node, where F is additionally *boxed*[4]. The *principal port* of the box

[4] The name CBN comes from the (en)*closure* of each argument by a box which can then be passed around without evaluating inside the box.

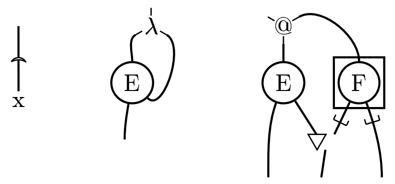


Fig. 2. Call-by-name graph coding.

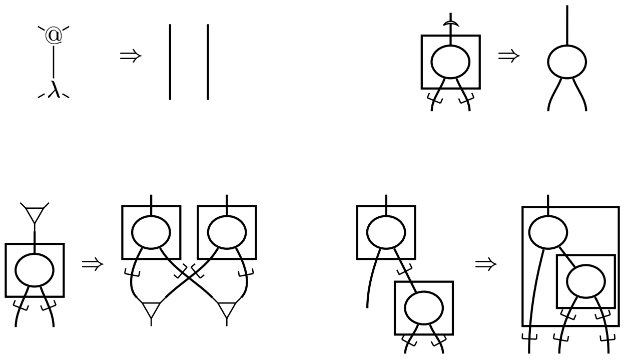


Fig. 3. Global reduction.

is located at the root of the graph coding F; each free variable v (coded by a croissant) in F, appearing as an *auxiliary port* of that box, is equipped with a bracket node that absorbs any box bound to v inside the box for F. Finally, if E and F share a free variable x, the two ports for x in their respective codings are combined with a sharing node.

We now describe three graph reduction schemes—*global*, *insular*, and *optimal.* Global and insular reductions are almost identical, and produce the same head normal forms. Insular and optimal reductions share the same context semantics on the *fragment* of the semantics used to compute readback. Combining these observations, we deduce that readback on optimally reduced graphs produces the head normal forms of global reduction—the crux of a correctness proof.

Global reduction (Figure 3) is the easiest to understand: $E \rightarrow_\beta E'$ iff $G_E \rhd_{\mathsf{G}} G_{E'}$. Define a *global box* to be a box together with the brackets *glued* to its free variable ports. A *global β-step* annihilates a λ-@ pair, and then propagates the (boxed) argument via *global duplications* and *global absorptions* to the variable occurrences, each a croissant that *globally opens* each propagated box.

Insular reduction (Figure 4) resembles global reduction, except that brackets are detached from boxes, and croissants and brackets do not vanish when a box is opened. An *insular β-step* annihilates a λ-@ pair, and then pushes sharing, bracket, and croissant nodes as far as possible. Sharing nodes duplicate boxes, but not brackets; croissants and brackets open or add a box to an existing box, propagating to free variable ports; and two boxes adjacent (via a graph edge) can *fuse* along their common boundary.

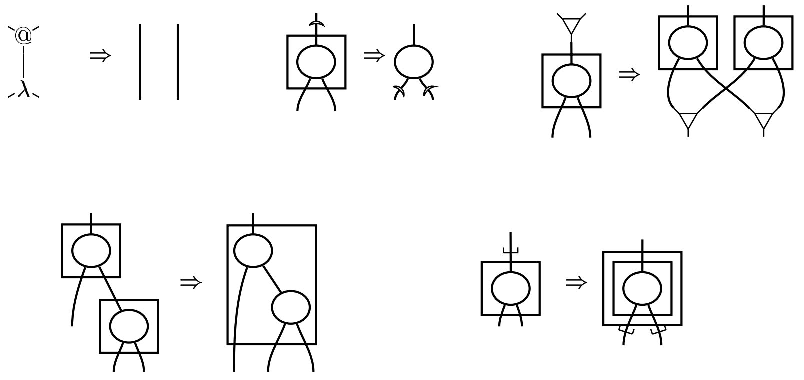


Fig. 4. Insular reduction.

Lemma 1. *Take a λ-term E and code it up as G_E. Perform leftmost-outermost* global *β-steps on G_E, deriving G'; also perform the same leftmost-outermost* insular *β-steps on G_E, deriving G''. Now erase all boxes, croissants, and brackets in G' and G''—they are now identical.*

Optimal reduction (Figure 5) looks totally different from global or insular reduction, since it has no global structure (boxes) in its reduction rules. (For those who object to globalization, here is the perfect interpreter: it is entirely local.) The initial coding G_E gives every graph node (@, λ, sharing, croissant, bracket) a *level*—the number of enclosing boxes. Then a croissant "opens" a boxed node by *decreasing* its level; a bracket "boxes" a node by *increasing* its level. In this way, the creation, opening, and sharing of boxes is done *incrementally.* Graphs quickly become inscrutable and do not look like λ-terms. But once we understand their *context semantics*, the resemblance to insular reduction will return.

Context semantics for sharing. To implement a semantics for sharing, the *wires* of graphs for linear λ-calculus are generalized to *buses* of wires. A *context* is no longer a string, but a vector $\langle\!\langle \chi_1, \ldots, \chi_n, \circ^+ \rangle\!\rangle$ where datum χ_i travels on the ith wire of the bus. First we explain how contexts are modified by graph nodes. Then we explain what contexts mean—what they say about λ-terms, reduction, and sharing.

How: A graph node at *level i* modifies χ_i. The functions below respectively describe how a context is changed by a function (@ and λ), sharing, croissant, and bracket node, where the context is input to an *auxiliary* port of the graph node. For example, $\mathbf{f}_{i,\circ}$ pushes a $\circ$ token on the ith wire, describing the transformation of a context approaching the $\circ$ port of an @- or λ node. Sharing nodes are made from similar hardware: we push tokens L and R instead[5]. A croissant with level i, representing an occurrence of variable v, is interpreted semantically as

[5] We could use $\circ$ and $\bullet$ instead of L and R, but we choose to increase "type safety" and make the contexts more readable.

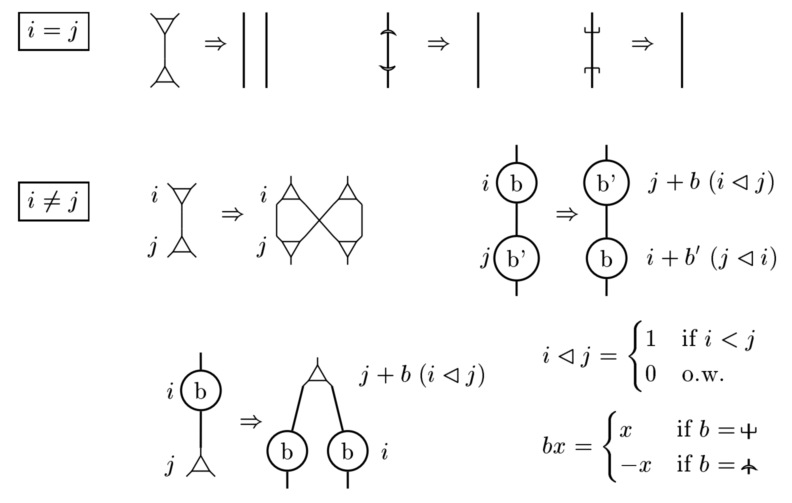

$i = j$
$i \neq j$
i
j
b
b'
$j + b\ (i \lhd j)$
$i + b'\ (j \lhd i)$
$i \lhd j = \begin{cases} 1 & \text{if } i < j \\ 0 & \text{o.w.} \end{cases}$
$bx = \begin{cases} x & \text{if } b = \\ -x & \text{if } b = \end{cases}$

Fig. 5. Optimal graph reduction.

$\mathbf{c}_{i,v}$, adding a "wire" to the context holding token v[6]. A bracket at level i is interpreted as $\mathbf{b}_i$, pairing two values onto one wire.

$$\begin{aligned}
\mathbf{f}_{i,t}\, \langle\!\langle \chi_1, \ldots, \chi_n, \circ^+ \rangle\!\rangle &= \langle\!\langle \chi_1, \ldots, \chi_{i-1}, t\chi_i, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle && [t \in \{\circ, \bullet\}] \\
\mathbf{s}_{i,t}\, \langle\!\langle \chi_1, \ldots, \chi_n, \circ^+ \rangle\!\rangle &= \langle\!\langle \chi_1, \ldots, \chi_{i-1}, t\chi_i, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle && [t \in \{\textsc{l}, \textsc{r}\}] \\
\mathbf{c}_{i,v}\, \langle\!\langle \chi_1, \ldots, \chi_n, \circ^+ \rangle\!\rangle &= \langle\!\langle \chi_1, \ldots, \chi_{i-1}, v, \chi_i, \ldots, \chi_n, \circ^+ \rangle\!\rangle \\
\mathbf{b}_{i}\, \langle\!\langle \chi_1, \ldots, \chi_n, \circ^+ \rangle\!\rangle &= \langle\!\langle \chi_1, \ldots, \chi_{i-1}, \langle \chi_i, \chi_{i+1} \rangle, \chi_{i+2}, \ldots, \chi_n, \circ^+ \rangle\!\rangle
\end{aligned}$$

The above functions *add* structure to a context; when a context encounters a croissant or bracket node at its *principal* port, structure is *consumed:*

$$\begin{aligned}
&\mathbf{c}_{i,v}^{-1}\, \langle\!\langle \chi_1, \ldots, \chi_{i-1}, v, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle = \langle\!\langle \chi_1, \ldots, \chi_{i-1}, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle \\
&\mathbf{b}_{i}^{-1}\, \langle\!\langle \chi_1, \ldots, \chi_{i-1}, \langle \chi_i, \chi_{i+1} \rangle, \chi_{i+2}, \ldots, \chi_n, \circ^+ \rangle\!\rangle \\
&\qquad = \langle\!\langle \chi_1, \ldots, \chi_{i-1}, \chi_i, \chi_{i+1}, \chi_{i+2}, \ldots, \chi_n, \circ^+ \rangle\!\rangle
\end{aligned}$$

At the principal port of a function (sharing) node, the path is routed to the left or right, in addition to the change of context—left if $t = \circ$ ($t = \textsc{l}$), and right if $t = \bullet$ ($t = \textsc{r}$), and token t is removed:

$$\begin{aligned}
\mathbf{f}_{i}^{-1}\, \langle\!\langle \chi_1, \ldots, \chi_{i-1}, t\chi_i, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle &= \langle\!\langle \chi_1, \ldots, \chi_{i-1}, \chi_i, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle \\
\mathbf{s}_{i}^{-1}\, \langle\!\langle \chi_1, \ldots, \chi_{i-1}, t\chi_i, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle &= \langle\!\langle \chi_1, \ldots, \chi_{i-1}, \chi_i, \chi_{i+1}, \ldots, \chi_n, \circ^+ \rangle\!\rangle
\end{aligned}$$

Why: What does all this stuff *mean?* A context can now identify a head variable not only by its binders and arguments ("x, λ-bound at ..., which is

[6] Again, v could be replaced with the null symbol ϵ, but we want to identify each croissant with the variable it represents.

the mth argument of head variable y, which is λ-bound at $\ldots$, which is the nth argument of head variable z, $\ldots$")—but also by its *occurrence*: which x is it? To provide this information, we need to know something about the *history of the computation* leading to the normal form. Context semantics provides this history.

Consider a context $\langle\!\langle \chi_1, \ldots, \chi_n, \circ^+ \rangle\!\rangle$ input at a port π of a graph[7]. If π is the root, each χ_{2i+1} traces a path along a chain of λ-binders, each χ_{2i} traces a path along a chain of applications, and if n is even (odd), it denotes an Opponent (Player) move that is input to (output from) the graph, identifying a subterm (head variable). Dually, when π is a free variable port, each χ_{2i+1} traces along applications, χ_{2i} along λ-binders, and and if n is odd (even), it denotes an Opponent (Player) move that is output from (input to) the graph. These parities change because at the root, the context enters with the orientation of a continuation, and at a free variable, with that of an expression.

Write each χ_i as $\overline{k}\sigma$, where $\overline{k}$ is a *numeral* $\circ^k\bullet$, and σ is a *sharing identifier*. The tokens from numerals are used by λ- and @-nodes, as in the linear case. Let V be a set of variables including those occurring in a λ-term; the sharing identifiers σ are generated from the grammar $\sigma \to V \mid \mathrm{L}\sigma \mid \mathrm{R}\langle\sigma, \sigma\rangle \mid \langle\sigma, \sigma\rangle$. Each sharing identifier σ explains how variable $\nu(\sigma)$ is shared, where $\nu(x) = x$, and $\nu(\mathrm{L}\sigma) = \nu(\mathrm{R}\langle\sigma', \sigma\rangle) = \nu(\langle\sigma', \sigma\rangle) = \nu(\sigma)$.

What do the sharing identifiers mean? A variable is the simplest identifier: in $\lambda x.x$, the initial Op-move $\langle\!\langle \circ^+ \rangle\!\rangle$ gives the Pl-response $\langle\!\langle \bullet x, \circ^+ \rangle\!\rangle$. In $\lambda x.xx$, the Player would respond $\langle\!\langle \bullet \mathrm{L}x, \circ^+ \rangle\!\rangle$—$\mathrm{L}x$ identifies the left occurrence. When the Opponent asks for the head variable of the argument of the head variable of $\lambda x.xx$, coded as the Op-move[8] $\langle\!\langle \bullet \mathrm{L}x, \bullet\alpha, \circ^+ \rangle\!\rangle$, the Player responds with $\langle\!\langle \bullet \mathrm{R}\langle\alpha, x\rangle, \circ^+ \rangle\!\rangle$—the *right* occurrence of x, occurring in a box that was "bound to" α. Finally, in $\lambda y.xy$, we have $\mathsf{Op}_1 = \langle\!\langle \circ^+ \rangle\!\rangle$ at the root, $\mathsf{Pl}_1 = \langle\!\langle x, \circ^+ \rangle\!\rangle$ at π_x, $\mathsf{Op}_2 = \langle\!\langle x, \bullet\alpha, \circ^+ \rangle\!\rangle$ at π_x, and $\mathsf{Pl}_2 = \langle\!\langle \bullet\langle\alpha, y\rangle, \circ^+ \rangle\!\rangle$ at the root—the sharing identifier $\langle\alpha, y\rangle$ says y is in a box "bound to" α, but not shared. In summary: σ represents the sharing of $\nu(\sigma)$. When $\sigma = \langle\sigma', \sigma''\rangle$ or $\mathrm{R}\langle\sigma', \sigma''\rangle$, the σ' describes the *external* sharing of the occurrence of the box containing the occurrence of $\nu(\sigma)$, and σ'' describes the *internal* sharing of $\nu(\sigma)$ inside the box. Check your intuitions with the following examples of contexts input and output (all at the root) to simple λ-terms; since all contexts have the form $\langle\!\langle \ldots, \circ^+ \rangle\!\rangle$—where the $\circ^+$ represents a "call" to a subterm—we henceforth eliminate its mention.

$\underline{\lambda x.\lambda y.xy}$**:** Then $\mathsf{Op}_1 = \langle\!\langle\, \rangle\!\rangle$, $\mathsf{Pl}_1 = \langle\!\langle \bullet x \rangle\!\rangle$ ("the head variable is x, bound"), $\mathsf{Op}_2 = \langle\!\langle \bullet x, \bullet\alpha \rangle\!\rangle$ ("the first argument of x—which the Opponent 'binds' to α—what's its head variable?"), $\mathsf{Pl}_2 = \langle\!\langle \circ \bullet \langle\alpha, y\rangle \rangle\!\rangle$ ("y, free in a box bound to α").

$\underline{\lambda x.x(\lambda y.y)}$**:** Again $\mathsf{Op}_1 = \langle\!\langle\, \rangle\!\rangle$, $\mathsf{Pl}_1 = \langle\!\langle \bullet x \rangle\!\rangle$ and $\mathsf{Op}_2 = \langle\!\langle \bullet x, \bullet\alpha \rangle\!\rangle$; but now $\mathsf{Pl}_2 = \langle\!\langle \bullet x, \bullet\alpha, \bullet y \rangle\!\rangle$ (contrast with the previous example).

[7] For now, imagine the graph is of a normal λ-term.

[8] Literally, $\alpha \equiv \epsilon$, but I include this notation to identify the Opponent's move in the context. Think of the Opponent as a projection function with head variable identified by α.

$\lambda x.x(\lambda y.xy)$: $\mathsf{Op}_1 = \langle\!\langle\,\rangle\!\rangle$, $\mathsf{Pl}_1 = \langle\!\langle \bullet \mathrm{L}x \rangle\!\rangle$, $\mathsf{Op}_2 = \langle\!\langle \bullet \mathrm{L}x, \bullet\alpha_0 \rangle\!\rangle$, $\mathsf{Pl}_2 = \langle\!\langle \bullet \mathrm{R}\langle \alpha_0, x\rangle \rangle\!\rangle$, $\mathsf{Op}_3 = \langle\!\langle \bullet \mathrm{R}\langle \alpha_0, x\rangle, \bullet\alpha_1 \rangle\!\rangle$, $\mathsf{Pl}_3 = \langle\!\langle \bullet \mathrm{L}x, \bullet\alpha_0, \bullet\langle \alpha_1, y\rangle \rangle\!\rangle$.

$\lambda x.\lambda y.\lambda z.\lambda w.x(y(zw))$: $\mathsf{Op}_1 = \langle\!\langle\,\rangle\!\rangle$, $\mathsf{Pl}_1 = \langle\!\langle \bullet x \rangle\!\rangle$, $\mathsf{Op}_2 = \langle\!\langle \bullet x, \bullet\alpha_1 \rangle\!\rangle$, $\mathsf{Pl}_2 = \langle\!\langle \circ \bullet \langle \alpha_1, y\rangle \rangle\!\rangle$, $\mathsf{Op}_3 = \langle\!\langle \circ \bullet \langle \alpha_1, y\rangle, \bullet\alpha_2 \rangle\!\rangle$, $\mathsf{Pl}_3 = \langle\!\langle \circ\circ\bullet \langle \alpha_1, \langle \alpha_2, z\rangle\rangle \rangle\!\rangle$, $\mathsf{Op}_4 = \langle\!\langle \circ\circ\bullet \langle \alpha_1, \langle \alpha_2, z\rangle\rangle, \bullet\alpha_3 \rangle\!\rangle$, $\mathsf{Pl}_4 = \langle\!\langle \circ\circ\circ\bullet \langle \alpha_1, \langle \alpha_2, \langle \alpha_3, w\rangle\rangle\rangle \rangle\!\rangle$.

Exercise: reading back the normal form of a normal λ-term. Given a *closed* λ-term $N \equiv \lambda x_0.\cdots\lambda x_p.x_i N_0 \cdots N_q$ in *normal form,* how do we read back N from the context semantics of G_N? We mimic the procedure introduced for linear λ-calculus, using the refined idea of contexts described above. Recall Op-moves identify subterms, and Pl-moves identify head variables; an Op-move has the form $O = \langle\!\langle \ell_1\sigma_1, a_1\alpha_1, \ldots, \ell_n\sigma_n, a_n\alpha_n \rangle\!\rangle$, and a Pl-move has the form $P = \langle\!\langle \ell_1\sigma_1, a_1\alpha_1, \ldots, \ell_n\sigma_n, a_n\alpha_n, \ell_{n+1}\sigma_{n+1} \rangle\!\rangle$, where the ℓ_i and a_i are numerals. The Opponent generates new moves by taking a Pl-move (coding a head variable v), and adding $\overline{k}\alpha$ just before the $\circ^+$ component ("what is the head variable of the kth argument of v?"). The α marks the occurrence of a projection variable from the Opponent.

Follow the path from the root to a subterm defined by a context: $\ell_1 = \circ^i\bullet$ traces through the λ-binders at the top level, reaching the head variable x_i of N. Then σ_1 represents *some* occurrence of x_i, and that occurrence (marked in the graph by a croissant node) is at *level* $d = \partial(\sigma_1)$, where $\partial(x) = 0$, $\partial(\alpha_j) = 1$, $\partial(\mathrm{L}\sigma) = \partial(\sigma)$, and $\partial(\mathrm{R}\langle\sigma',\sigma\rangle) = \partial(\langle\sigma',\sigma\rangle) = \partial(\sigma') + \partial(\sigma)$. The path then unpacks $\alpha'_1, \ldots, \alpha'_d$ from σ_1, and consumes the token x_i. Next $a_1 = \circ^j\bullet$ is consumed to trace through a chain of @-nodes, en route to the graph of subterm N_j, while placing α_2 from the context on the $(d+1)$st wire. The explanation continues inductively on $\langle\!\langle \ell_2\sigma_2, a_2\alpha_2, \ldots, \ell_n\sigma_n, a_n\alpha_n \rangle\!\rangle$.

For the head variable identified by the Player in move P above, $\ell_{n+1}\sigma_{n+1}$ gives the numeral ℓ_{n+1} coding the lexical address of the head variable $v = \nu(\sigma_{n+1})$ in the subterm N' containing v's *binder*, and $\partial(\sigma_{n+1})$ gives the *distance* (number of enclosing boxes) separating that binder from the *occurrence* of v in the enclosed subterm N''. The remaining information in move P gives the address of N'. Moreover, σ_{n+1} identifies the occurrences of the Op-variables bound to each of the boxes separating the occurrence of v from its binder.

Readback does not require any information from the σ_i, *except* that $\partial(\sigma_i)$ be correct, since it indicates how many boxes have to be entered to get to the head variable in question. The rest of the information in σ_i tells what variable occurrences $\boldsymbol{v}$ were bound to these boxes during reduction, and what variable occurrences $\boldsymbol{v'}$ were bound to boxes containing $\boldsymbol{v}$, and so on. Now consider readback on a term that is strongly normalizable: use *insular reduction* to compute its normal form.

Proposition 4. *In an insular β-step, when a variable x is bound to a box* B*, the sharing, bracket, and—in particular—croissant nodes at the* occurrences *of x move to the free variable ports of the copies of* B *produced during the reduction. In that latter position, they* do not change the level *of head variables; equivalently, their contribution to sharing identifiers σ in the context semantics code the correct depth.*

As a consequence, when entering the principal port of a box, the context resembles that which is found in readback of a normal λ-term.

Proposition 5. *When the principal port of a box at level d is encountered on a path, the first d wires of the context hold $\alpha'_1, \cdots \alpha'_d$, all from projection functions of the Opponent.*

In other words: paths from head variables to the root pack sharing information into the context, which is *unpacked* (in the same graph location!) on return paths to subterms. It does not matter *what* this information is, as long as it has the correct depth. Recall Lemma 1, and let G' and G'' be the respective graphs derived by global and insular reduction to the normal form of the same λ-term: then G', G'' only differ in the accumulation of bracket and croissant nodes at free variable ports of boxes. In G'', these *control nodes* are the debris of β-contraction—but for mere readback of the λ-term, they are only sound and fury, signifying nothing.

Lemma 2. *Let G_E be the graph coding a λ-term E. Given the context semantics of the* normalized *graph produced by either global or insular reduction of G_E, the readback algorithm computes the normal form of E.*

Readback: the general case and optimal reduction. However, both global and insular reduction can *change* the context semantics. For example, in $(\lambda z.\lambda w.wzz)(xy)$, the context semantics includes paths from π_x to π_y that do not include sharing information (i.e., the use of L or R). But in the context semantics of global or insular normal forms, from which we can read back $xy(xy)$, this information is required, since a sharing node is found on every path from π_x to π_y. So if the context semantics changes, what relation is there between the context semantics of the graph at the start and conclusion of reduction? How do we know that we are reading back the normal form of the right term? The solution is to recognize that insular reduction does not change the fragment of the context semantics that is used for readback. Readback traces paths in a *proper* manner, a straightforward idea that we now elaborate:

Definition 1. *A* proper interaction *with a box H at level k is a sequence of contexts $I = \{o_1, p_1, \ldots, o_n, p_n\}$ that alternately enter and exit the ports of H, where*

1. *Let $1 \le r \le k$; the rth component of every context in I is a sharing identifier. Moreover, any two contexts in I have the same such identifier in the rth component.*
2. *They are consistent with the context semantics: the input of o_i followed by the output of p_i is determined by the functions defined by nodes along the path from o_i to p_i;*
3. *o_1 is input at the principal port (root) of the box, and each o_{i+1} is input at the port where p_i was output;*
4. *If the port of p_i and o_{i+1} is not the root, then their $(k+1)$st wires contain the same sharing identifier (i.e., they are talking about the same variable occurrence in the box); and*
5. *Any interactions with boxes contained in H are also proper.*

Lemma 3. *The context semantics defines an interaction with an implicit "box" that encloses the entire graph. In readback on an insular or global normal form, this interaction is proper. Furthermore, an interaction with a box is proper iff an interaction with any* insular *graph reduction of that box is proper. (The latter assertion is false for global reduction.)*

The proof of this lemma is a slightly tedious but straightforward induction on reduction sequences, where we check that each insular graph reduction rule preserves the requisite properties. Since readback on an (insular) normalized graph is a proper interaction, we conclude:

Theorem 1. *Let G_E be the graph coding a λ-term E. Given the context semantics for G_E, the readback algorithm produces the normal form of E. Since optimal reduction leaves the* entire *context semantics invariant, readback on optimally reduced graphs is correct.*

We remark that in readback of an optimally reduced graph, we use the graph to *compute* the necessary fragment of the context semantics, rather than being presented with it extensionally. Observe that this theorem can be strengthened so that E need not have a normal form; in this case, the readback algorithm computes the Böhm tree.

4 Types

We can *type* terms by attaching data types that are linear logic formulas to oriented graph edges. A λ-node (@-node) has an outgoing (ingoing) type $\alpha \multimap \beta$ on its principal root (function) port, an outgoing (ingoing) type α on its auxiliary parameter (argument) port, and an ingoing (outgoing) type β on its auxiliary body (continuation) port. A box has ingoing types $!\alpha_1, \ldots, !\alpha_n$ on its auxiliary ports, and an outgoing type $!\beta$ on its principal port. A sharing node has $!\alpha$ going in at its principal port, and $!\alpha$ going out at its auxiliary ports. A croissant (bracket) has $!\alpha$ ($!\alpha$) going in at its principal port, and α ($!!\alpha$) at its auxiliary port. The CBN coding assumes arguments are always boxed, so simple types τ are compiled into linear logic types $[\tau]$ as $[A] = A$ and $[\alpha \to \beta] = ![\alpha] \multimap [\beta]$. These types are preserved by global and insular reduction, but not by optimal reduction[9].

In the bus system, a subterm of type $! \cdots !\alpha$ (m !s) must be enclosed in m boxes, and each box has a dedicated wire on the bus, holding information about the *sharing* of that box—what variable occurrence is bound to it? From these contexts we can read part of the computation history. The information flow on each wire clarifies how context semantics can be interpreted as a kind of flow analysis, describing the fine structure of games in the style of McCusker's informal presentation [2]. The Curry-Howard correspondence is reinterpreted using the games slogan: types are *arenas* where games can be played (Player: "I am

[9] Consider the sharing of a function with type $!\mathsf{Int} \multimap \mathsf{Int}$. By duplicating the λ-node, we then need to share the output of type Int—without a !, which is type incorrect.

thinking of a term of type τ"; Opponent: "What is its head variable?"...), and a λ-term of that type represents a winning Pl-strategy[10]. In the typed framework we can dispense with the $\circ^+$-notation, using the types to insert the exact number of needed tokens. For example, the Pl-strategy 3 has $\mathsf{Op}_1 = ?$, $\mathsf{Pl}_1 = 3$; the strategy $\mathsf{succ} : \mathsf{Int} \multimap \mathsf{Int}$ has (for each n) $\mathsf{Op}_1 = \circ ?$, $\mathsf{Pl}_1 = \bullet ?$, $\mathsf{Op}_2 = \bullet n$, $\mathsf{Pl}_2 = \circ(n+1)$; and $\mathsf{succ}' : !\mathsf{Int} \multimap \mathsf{Int}$ has $\mathsf{Op}_1 = \langle\!\langle \circ ? \rangle\!\rangle$, $\mathsf{Pl}_1 = \langle\!\langle \bullet\alpha, ? \rangle\!\rangle$, $\mathsf{Op}_2 = \langle\!\langle \bullet\alpha, n \rangle\!\rangle$, $\mathsf{Pl}_2 = \langle\!\langle \circ(n+1) \rangle\!\rangle$. Observe that a wire of type Int carries requests (of type $\mathsf{Int}^\perp$) for an integer data in one direction, and supplies an integer (Int) in the other direction. An edge of type $!\mathsf{Int}$ also has a dedicated "sharing wire" identifying which shareholder wants the data; an edge of type $!!\mathsf{Int}$ also explains how the shareholder is also shared. A free variable $x : !\mathsf{Int}$ occurring in a box has an *internal* sharing wire (how is x shared in the box?), and an *external* sharing wire (how is the box shared?) that is "threaded" through the entire box. Upon exiting at x's port, these wires are paired by the bracket. When a !!-type becomes a !-type inside a box without transformation by any graph node, only the type information internal to the box is being shown.

Consider the example $(\lambda f : !(!\mathsf{Int} \multimap \mathsf{Int}).f(f(!3)))\mathsf{succ}' : \mathsf{Int}$ (annotated using the CBN translation), where $\mathsf{Op}_1 = ?$ and $\mathsf{Pl}_1 = 5$—but what are the calls to the *boxed* version of succ' during the computation? The first call is $\mathsf{C}_1 = \langle\!\langle \mathrm{L}f, \circ ? \rangle\!\rangle$ (what's the output?), and succ' responds $\mathsf{S}_1 = \langle\!\langle \mathrm{L}f, \bullet\alpha, ? \rangle\!\rangle$ (what's the input?); then $\mathsf{C}_2 = \langle\!\langle \mathrm{R}\langle \alpha, f \rangle, \circ ? \rangle\!\rangle$ (what's the output?—but for a *new* call site), $\mathsf{S}_2 = \langle\!\langle \mathrm{R}\langle \alpha, f \rangle, \bullet\alpha, ? \rangle\!\rangle$ (again, what's the input?), $\mathsf{C}_3 = \langle\!\langle \mathrm{R}\langle \alpha, f \rangle, \bullet\alpha, 3 \rangle\!\rangle$ (the input is 3), $\mathsf{S}_3 = \langle\!\langle \mathrm{R}\langle \alpha, f \rangle, \circ 4 \rangle\!\rangle$ (the output is 4), $\mathsf{C}_4 = \langle\!\langle \mathrm{L}f, \bullet\alpha, 4 \rangle\!\rangle$ (the input is 4), $\mathsf{S}_4 = \langle\!\langle \mathrm{L}f, \circ 5 \rangle\!\rangle$ (the output is 5). Each wire holds type-dedicated information; for example, in S_3, $\mathrm{R}\langle \alpha, f \rangle$ has type $!(!\mathsf{Int} \multimap \mathsf{Int})$ (sharing the function), and $\circ 4$ has type $!\mathsf{Int} \multimap \mathsf{Int}$; the tag $\circ$ indicates an injection from Int. Dually, in C_4, $\mathrm{L}f : [!(!\mathsf{Int} \multimap \mathsf{Int})]^\perp$, $\bullet\alpha : (!\mathsf{Int} \multimap \mathsf{Int})^\perp = !\mathsf{Int} \otimes \mathsf{Int}^\perp$ (where $\bullet$ is the injection from $!\mathsf{Int}$), and $3 : (\mathsf{Int} \multimap \perp)^\perp = \mathsf{Int}$. In an initial coding, a bus for an edge of type α has wires of type $\alpha, \phi(\alpha), \phi^2(\alpha), \ldots$, where $\phi^{n+1}(\alpha) = \perp$ for a base type, $\phi^{n+1}(!\alpha) = \phi^n(\alpha)$, and $\phi^{n+1}(\alpha \multimap \beta) = \phi^n(\alpha) \multimap \phi^n(\beta)$. This elaborated typing can be shown to be preserved by optimal reduction.

5 Labelled $\boldsymbol{\lambda}$-Calculus *à la Lévy* and Paths

Give every application and abstraction a unique *atomic* label, and define *labelled β-reduction* as $(\lambda x.E)^\ell F \rhd_{\mathsf{lab}} ([F^{\underline{\ell}}/x]E)^\ell$. For example, consider *labelled reduction* of $(\lambda x.xx)(\lambda x.xx)$ (see Figure 6):

$$\begin{aligned}
&((\lambda x.(x^1 x^2)^3)^4 (\lambda x(x^5 x^6)^7)^8)^9 \\
&\quad \rhd_{\mathsf{lab}} ((\lambda x.(x^5 x^6)^7)^{8\underline{4}1} (\lambda x(x^5 x^6)^7)^{8\underline{4}2})^{349} \\
&\quad \rhd_{\mathsf{lab}} ((\lambda x.(x^5 x^6)^7)^{8\underline{4}2\underline{8\underline{4}1}5} (\lambda x.(x^5 x^6)^7)^{8\underline{4}2\underline{8\underline{4}1}6})^{78\underline{4}1349}
\end{aligned}$$

[10] In game semantics, the *moves* of game $A \otimes B$ are defined as the disjoint union of A and B; the $\bullet$ and $\circ$ tokens of context semantics are what implement this disjoint union. In this spirit, context semantics realizes a sort of "machine language" for implementing games.

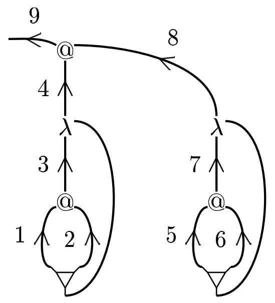


Fig. 6. Labelled reduction and paths.

Now interpret underlining as *reversal*, so $\underline{\underline{\ell}} = \ell$ and $\underline{\ell\ell'} = \underline{\ell'}\,\underline{\ell}$. For example, we have $8\underline{4}2\underline{8\underline{4}1}5 = 8\underline{4}2\underline{1}4\underline{8}5$. Observe that this label on the function in the last application above, describes a path between an @- and λ-node in the initial graph coding of $(\lambda x.xx)(\lambda x.xx)$—read underlined atomic labels by *reversing* the orientation of the graph edge. This connection between labels and paths has been studied in [5]; we add some intuitions. First, except for K-redexes, which discard term structure, the history of the computation is coded in the labels: we can effectively run the reductions backwards. But the same is not true of context semantics—we cannot reconstruct the initial term. For example, take $((\lambda x.(x^1x^2)^3)^4(\lambda y.y^5)^6)^7$, which reduces to $(\lambda y.y^5)^\ell$ where $\ell = 6\underline{4}2\underline{6\underline{4}1}56\underline{4}1347$—try tracing this path, for fun. But the context semantics of this term is just that of $\lambda y.y$—so what is it about reductions that lets ℓ get *forgotten* by the context semantics?

Just as optimal reductions commute bracket and croissant nodes through @ and λ-nodes, rewrite $(U^\alpha V)$ as $(UV^{\underline{\alpha}})^\alpha$ and $(\lambda x.E)^\alpha$ as $\lambda x.E^\alpha[x^{\underline{\alpha}}/x]$. Forget all atomic labels except those marking variable occurrences. Then the labelling of $(\lambda y.y^5)^\ell$ gets rewritten to $(\lambda y.y^5)^{2\underline{1}51}$ and again to $\lambda y.y^{\underline{2\underline{1}51}52\underline{1}51}$. By adding rules that commute and annihilate atomic labels, imitating those for optimal reduction, we should be able to deduce that the outer label self-annihilates.

6 Conclusions and Open Problems

This largely tutorial paper represents an important part of research: to *re-search* some of what has already been found, and to know it better. Context semantics is an expressive foundation for static program analysis. We have given a simple correctness proof, explaining why *readback* of the semantics—unchanged by optimal reduction—yields the normal form. What *fragments* of this semantics correspond to *tractable* schemes for static program analysis? It would be nice to tie this semantics more closely to that of games, so that an Opponent could ask questions about aspects of the computation history, i.e., "what variable got bound to the box containing the head variable?". These sorts of questions suggest the possibility of full abstraction theorems that are sensitive to *sharing*—that two $\beta\eta$-equivalent terms are distinguishable if they share applications differently. The informal typing of individual wires in Section 4 could be a step in providing a semantics of optimal reduction where the nodes in "negative" position are explained properly. The relation of labels to semantics, briefly discussed in Section 5, deserves a clearer and more detailed explanation. The box-croissant-bracket metaphor, as explained in [3], is just a *comonad*: the box is a functor, and the croissant and bracket are the *unit* and *bind* described in [14]. Why not use graph reduction, then, as a means of implementing monads? For example, *state* can be represented by the !-dual functor ?, so in a game for $?\alpha$, the Opponent could query the state as well as the type α. There is more of this story to be told.

Dedicated in memory of my father, Theodore Mairson (1919–2002)

Acknowledgements

Thanks to many people I have discussed these ideas with over several years, including especially Alan Bawden, Jakov Kucan, Julia Lawall, Jean-Jacques Lévy, Peter Neergaard, and Luca Roversi. Special thanks to Peter who did the figures for me at the last minute.

References

1. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163(2):409–470, Dec. 2000.
2. S. Abramsky and G. McCusker. Game semantics. In H. Schwichtenberg and U. Berger, editors, *Theoretical Foundations of Computer Graphics and CAD*, volume 165 of *NATO ASI*, pages 307–325. Springer-Verlag, 1999.
3. A. Asperti. Linear logic, comonads and optimal reductions. *Fundamentae Informaticae*, 22:3–22, 1995.
4. A. Asperti and S. Guerrini. *The Optimal Implementation of Functional Programming Languages.* Cambridge University Press, 1998.
5. A. Asperti and C. Laneve. Paths, computations and labels in the λ-calculus. *Theoretical Computer Science*, 142(2):277–297, 15 May 1995.
6. A. Asperti and H. G. Mairson. Parallel beta reduction is not elementary recursive. *Information and Computation*, 170:49–80, 2001.
7. J.-Y. Girard. Linear logic. *Theoretical Computer Science*, 50, 1987.
8. J.-Y. Girard. Geometry of interaction I: Interpretation of system F. In C. Bonotto, R. Ferro, S. Valentini, and A. Zanardo, editors, *Logic Colloquium '88*, pages 221–260. North-Holland, 1989.
9. G. Gonthier, M. Abadi, and J.-J. Lévy. The geometry of optimal lambda reduction. In *Conference record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: papers presented at the symposium, Albuquerque, New Mexico, January 19–22, 1992*, pages 15–26, New York, NY, USA, 1992. ACM Press.
10. G. Gonthier, M. Abadi, and J.-J. Lévy. Linear logic without boxes. In *Proceedings 7th Annual IEEE Symp. on Logic in Computer Science, LICS'92, Santa Cruz, CA, USA, 22–25 June 1992*, pages 223–34. IEEE Computer Society Press, Los Alamitos, CA, 1992.

11. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II, and III. *Information and Computation*, 163(2):285–408, Dec. 2000.
12. J. Lamping. An algorithm for optimal lambda calculus reduction. In *POPL '90. Proceedings of the Seventeenth Annual ACM Symposium on Principles of Programming Languages, January 17–19, 1990, San Francisco, CA*, pages 16–30, New York, NY, USA, 1990. ACM Press.
13. J.-J. Lévy. *Réductions correctes et optimales dans le lambda-calcul.* PhD thesis, Université Paris 7, 1978. Thèse d'Etat.
14. P. Wadler. The essence of functional programming. In *Conference record of the Nineteenth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages: papers presented at the symposium, Albuquerque, New Mexico, January 19–22, 1992*, pages 1–14, New York, NY, USA, 1992. ACM Press.

Encoding Generic Judgments*

Dale Miller[1] and Alwen Tiu[2]

[1] INRIA-FUTURS and Ècole Polytechnique
dale.miller@inria.fr
[2] Pennsylvania State University and Ècole Polytechnique
tiu@cse.psu.edu

Abstract. The operational semantics of a computation system is often presented as inference rules or, equivalently, as logical theories. Specifications can be made more declarative and high-level if syntactic details concerning bound variables and substitutions are encoded directly into the logic using term-level abstractions (λ-abstraction) and proof-level abstractions (eigenvariables). When one wishes to reason about relations defined using term-level abstractions, *generic judgment* are generally required. Care must be taken, however, so that generic judgments are not uniformly handled using proof-level abstractions. Instead, we present a technique for encoding generic judgments and show two examples where generic judgments need to be treated at the term level: one example is an interpreter for Horn clauses extended with universal quantified bodies and the other example is that of the π-calculus.

1 Introduction

The operational semantics of a programming or specification language is often given in a relational style using inference rules following a small-step approach (a.k.a., structured operational semantic [Plo81]) or big-step approach (a.k.a. natural semantics [Kah87]). In either case, algebraic (first-order) terms can be used to encode the language being specified and the first-order theory of Horn can be used to formalize and interpret such semantic specifications. For example, consider the following natural semantics specification of a conditional expression for a functional programming language:

$$\frac{B \Downarrow \mathit{true} \qquad M \Downarrow V}{(\mathit{if}\ B\ M\ N) \Downarrow V} \qquad\qquad \frac{B \Downarrow \mathit{false} \qquad N \Downarrow V}{(\mathit{if}\ B\ M\ N) \Downarrow V}$$

These two inference figures can also be seen as two first-order Horn clauses:

$$\begin{array}{l} \forall B \forall M \forall N \forall V [B \Downarrow \mathit{true} \wedge M \Downarrow V \ \supset (\mathit{if}\ B\ M\ N) \Downarrow V] \\ \forall B \forall M \forall N \forall V [B \Downarrow \mathit{false} \wedge N \Downarrow V \ \supset (\mathit{if}\ B\ M\ N) \Downarrow V] \end{array}$$

Here, the down arrow is a non-logical, predicate symbol and an expression such as $N \Downarrow V$ is an atomic formula.

* An earlier draft of this paper appeared in MERLIN 2001 [Mil01].

Of course, once a specification is made, one might want to reason about it. For example, if these two rules are the only rules describing the evaluation of the conditional, then it should follow that if $(\mathit{if}\ B\ M\ M) \Downarrow V$ is provable then so is $M \Downarrow V$. In what logic can this be formalized and proved? For example, how might we prove the sequent

$$(\mathit{if}\ B\ M\ M) \Downarrow V \longrightarrow M \Downarrow V,$$

where B, M, and V are eigenvariables (universally quantified)? Since such a sequent contains no logical connectives, the standard sequent inference rules that introduce logical connective will not directly help here. One natural extension of the sequent calculus is then to add left and right introduction rules for atoms. Hallnäs and Schroeder-Heister [HSH91,SH93], Girard [Gir92], and more recently, McDowell and Miller [MM97,MM00] have all considered just such introduction rules for non-logical constants, using a notion of *definition*.

2 A Proof Theoretic Notion of Definitions

A *definition* is a finite collection of definition *clauses* of the form $\forall \bar{x}[H \triangleq B]$, where H is an atomic formula (the one being defined), every free variable of the formula B is also free in H, and all variables free in H are contained in the list $\bar{x}$ of variables. Since all free variables in H and B are universally quantified, we often leave these quantifiers implicit when displaying definition clauses. The atomic formula H is called the *head* of the clause, and the formula B is called the *body*. The symbol $\triangleq$ is used simply to indicate a definition clause: it is not a logical connective. The same predicate may occur in the head of multiple clauses of a definition: it is best to think of a definition as a mutually recursive definition of the predicates in the heads of the clauses. Let H, B, and $\bar{x}$ be restricted as above. If we assume further that the only logical connectives that may occur in B are $\exists$, $\top$, and $\wedge$, then we say that the formula $\forall \bar{x}[B \supset H]$ is a *Horn clause* and the definition clause $\forall \bar{x}[H \triangleq B]$ is a *Horn definition clause*.

Given a definition, the following two inference rules are used to introduce defined predicates. The right introduction rule is

$$\frac{\Gamma \longrightarrow B\theta}{\Gamma \longrightarrow A}\ \mathit{def}\mathcal{R}\ ,$$

provided that there is a clause $\forall \bar{x}[H \triangleq B]$ in the given definition such that A is equal to $H\theta$. The left-introduction rule is

$$\frac{\{B\theta, \Gamma\theta \longrightarrow C\theta \mid \theta \in \mathit{CSU}(A,H) \text{ for some clause } \forall \bar{x}[H \triangleq B]\}}{A, \Gamma \longrightarrow C}\ \mathit{def}\mathcal{L}\ ,$$

where the variables $\bar{x}$ are chosen (via α-conversion) to be distinct from the (eigen)variables free in the lower sequent of the rule. The set $\mathit{CSU}(A,H)$ denotes a complete set of unifiers for A and H: when the CSUs and definition are finite, this rule will have a finite number of premises. (A set S of unifiers for t and u is *complete* if for every unifier ρ of t and u there is a unifier $\theta \in S$ such that ρ is $\theta \circ \sigma$ for some substitution σ [Hue75].) There are many important situations where CSUs are not only finite but are also singleton (containing a most general unifier) whenever terms are unifiable: in particular, first-order unification and the *higher-order pattern* unification [Mil91a], where the application of functional variables are restricted to distinct bound variables.

We must also restrict the use of implication in the bodies of definition clauses, otherwise cut-elimination does not hold [SH92]. To that end we assume that each predicate symbol p in the language is associated with it a natural number $\mathrm{lvl}(p)$, the *level* of the predicate. We then extend the notion of level to formulas and derivations. Given a formula B, its *level* $\mathrm{lvl}(B)$ is defined as follows:

1. $\mathrm{lvl}(p\,\bar{t}) = \mathrm{lvl}(p)$
2. $\mathrm{lvl}(\bot) = \mathrm{lvl}(\top) = 0$
3. $\mathrm{lvl}(B \wedge C) = \mathrm{lvl}(B \vee C) = \max(\mathrm{lvl}(B), \mathrm{lvl}(C))$
4. $\mathrm{lvl}(B \supset C) = \max(\mathrm{lvl}(B) + 1, \mathrm{lvl}(C))$
5. $\mathrm{lvl}(\forall x.B) = \mathrm{lvl}(\exists x.B) = \mathrm{lvl}(B)$.

We now require that for every definition clause $\forall\bar{x}[p\,\bar{t} \stackrel{\triangle}{=} B]$, $\mathrm{lvl}(B) \leq \mathrm{lvl}(p)$. (If a definition is restricted to the Horn case, then this restriction is trivial to satisfy.) Cut-elimination for this use of definition within intuitionistic logic was proved in [McD97,MM00]. In fact, that proof was for a logic that also included a formulation of induction.

Definitions are a way to introduce logical equivalences so that we do not introduce into proof search meaningless cycles: for example, if our specification contains the equivalence $H \equiv B$, then when proving a sequent containing H, we could replace it with B, which could then be replaced with H, etc.

To illustrate the strengthening of logic that can result from adding definitions in this way, consider our first example sequent above. We first convert the two Horn clauses representing the evaluation rules for the conditional into the following definition clauses.

$$\begin{array}{l}(\mathit{if}\ B\ M\ N) \Downarrow V \stackrel{\triangle}{=} B \Downarrow \mathit{true} \wedge M \Downarrow V.\\ (\mathit{if}\ B\ M\ N) \Downarrow V \stackrel{\triangle}{=} B \Downarrow \mathit{false} \wedge N \Downarrow V.\end{array}$$

This sequent then has the following simple and immediate proof.

$$\dfrac{\dfrac{\dfrac{}{B \Downarrow \mathit{true}, M \Downarrow V \longrightarrow M \Downarrow V}\ \mathit{initial}}{B \Downarrow \mathit{true} \wedge M \Downarrow V \longrightarrow M \Downarrow V}\ \wedge L \qquad \dfrac{\dfrac{}{B \Downarrow \mathit{false}, M \Downarrow V \longrightarrow M \Downarrow V}\ \mathit{initial}}{B \Downarrow \mathit{false} \wedge M \Downarrow V \longrightarrow M \Downarrow V}\ \wedge L}{(\mathit{if}\ B\ M\ M) \Downarrow V \longrightarrow M \Downarrow V}\ \mathit{def}\mathcal{L}$$

In the paper [MMP01], the expressive strength of definitions was studied in greater depth. One example considered there involved attempting to capture the notion of simulation and bisimulation for labeled transition systems. In particular, assume that $P \xrightarrow{A} P'$ is defined via clauses to which are added the two

$$\begin{aligned}
\mathit{sim}\ P\ Q &\stackrel{\triangle}{=} \forall A \forall P'.\ P \xrightarrow{A} P' \supset \exists Q'.\ Q \xrightarrow{A} Q' \wedge \mathit{sim}\ P'\ Q' \\
\mathit{bisim}\ P\ Q &\stackrel{\triangle}{=} [\forall A \forall P'.P \xrightarrow{A} P' \supset \exists Q'.Q \xrightarrow{A} Q' \wedge \mathit{bisim}\ P'\ Q'] \wedge \\
&\qquad [\forall A \forall Q'.Q \xrightarrow{A} Q' \supset \exists P'.P \xrightarrow{A} P' \wedge \mathit{bisim}\ Q'\ P']
\end{aligned}$$

Fig. 1. Simulation and bisimulation as definitions.

clauses in Figure 1. These two clauses are a direct encoding of the closure conditions for simulation and bisimulation. (To satisfy the restrictions on levels, the level given to the predicate $\cdot \xrightarrow{\cdot} \cdot$ must be less than the level given to either predicate *sim* or *bisim*.) In [MMP01] it was proved that if the labeled transition system is finite (noetherian) then simulation and bisimulation coincided exactly with provability of $\mathit{sim}\ P\ Q$ and $\mathit{bisim}\ P\ Q$. Since provability characterizes the least fixed point and simulation and bisimulation are characterized using the greatest fixed point, the restriction to noetherian transition systems is necessary since noetherian guarantees that these two fixed points are the same.

In Section 4 we show how we can capture simulation and bisimulation for the (finite) π-calculus in a similar style.

3 λ-Tree Syntax and Generic Judgments

It is a common observation that first-order terms are not expressive enough to capture rich syntactic structures declaratively. In particular, such terms do not permit a direct encoding of the syntactic category of "abstraction" and the associated notions of α-conversion and substitution.

3.1 Syntactic Representation of Abstractions

The encoding style called *higher-order abstract syntax* [PE88] views such abstractions as functional expressions that rely on the full power of β-conversion in a typed λ-calculus setting to perform substitutions. The computer systems λProlog, Elf, Isabelle, and Coq, to name a few, all implement a form of HOAS and many earlier papers have appeared exploiting this style of syntactic representation [MN85,MN86,MN87,Pau89]. Since the earliest papers, however, there has been a tendency to consider richer λ-calculi as foundations for HOAS, moving away from the simply typed λ-calculus setting where it was first exploited. Trying to encode a syntactic category of abstraction by placing it within a rich function spaces can cause significant problems (undecidable unification, exotic terms, etc) that might seem rather inappropriate if one is only trying to develop a simple treatment of syntax.

The notion of λ-*tree syntax* [MP99,Mil00] was introduced to work around these complexities. Here, λ-abstractions are not general functions: they can only be applied to other, internally bound variables. Substitution of general values is not part of the equality theory in the λ-term syntax approach: it must be coded as a separate judgment via logic. This weaker approach notion of equality gives rise to L_λ unification (also, higher-order pattern unification) [Mil91a,Nip91], which is decidable and unary. The relationship between the λ-tree approach where abstractions are applied to only internally bound variable and HOAS where abstractions can be applied to general terms is rather similar to the distinctions made in the π-calculus between π_I, which only allows "internal mobility" [San96] and the full π-calculus, where "external mobility" is also allowed (via general substitutions). In Section 4, we will see that this comparison is not accidental.

3.2 When No Object-Level Abstractions Are Present

To help illustrate the special needs of reasoning about syntax that contains abstractions, we start with an example in which such abstractions are not present.

Consider the specification of an interpreter for object-level Horn clauses. We shall use the type o to denote meta-level logical expressions and obj to denote object-level logical expressions. The meta-logic denotes universal and existential quantification at type σ as $\forall_\sigma$ and $\exists_\sigma$ (both of type $(\sigma \to o) \to o$) and denotes conjunction and disjunction as $\wedge$ and $\supset$ (both of type $o \to o \to o$). The object-logic denotes universal and existential quantification by $\hat{\forall}_\sigma$ and $\hat{\exists}_\sigma$ (both of type $(\sigma \to obj) \to obj$), conjunction and implication by & and $\Rightarrow$ (both of type $obj \to obj \to obj$), and truth by $\hat{\top}$ (of type obj). Following usual conventions: type subscripts on quantifiers will often be dropped if they can be easily inferred or are not important, and if a quantifier is followed by a lambda abstraction, the lambda is dropped; eg, $\hat{\forall}\lambda x$ will be abbreviated as simply $\hat{\forall} x$.

To encode the provability relation for the object-logic, we exploit the completeness of goal-directed proofs for our object-logic [Mil90]. Provability can be specified as an interpreter using the following four (meta-level) predicates: provability is denoted by $\rhd$ and has type $obj \to o$, backchaining is denoted by the infix symbol $\lhd$ and has type $obj \to obj \to o$, *atomic* of type $obj \to o$ decides if an object-level formula is atomic, and *prog*, also of type $obj \to o$, decides if a formula is an object-level assumption (object-level logic program). The definition in Figure 2 is an interpreter for object-level Horn clause.

Completing the specification of the interpreter requires additional definition clauses for specifying what are atomic object-level formulas and what formulas constitute the object-level Horn clause specification. Examples of such clauses are given in Figure 3. Here, $\Downarrow$ has type $tm \to tm \to obj$, where tm is the type of the programming language for which evaluation is being specified. It is possible to now prove the sequent

$$\rhd(\mathit{if}\ B\ M\ M) \Downarrow V \longrightarrow \rhd M \Downarrow V$$

using the definition clauses in Figures 2 and 3. In fact, the proof of this sequent follows the same structure as the proof of this sequent when it is considered at the meta-level only (as in Section 2). That is, although the proof using the meta-level/object-level distinctions is more complex and detailed, no interesting information is uncovered by these additional complexities. The reason to present

$$\begin{array}{rlrl}
\triangleright \hat{\top} & \triangleq \top. & A \lhd A & \triangleq \mathit{atomic}\ A. \\
\triangleright(G \mathbin{\&} G') & \triangleq \triangleright G \wedge \triangleright G'. & (G \Rightarrow D) \lhd A & \triangleq D \lhd A \wedge \triangleright G. \\
\triangleright(\hat{\exists}_\sigma\, G) & \triangleq \exists_\sigma x. \triangleright(Gx) & (\hat{\forall}_\sigma\, D) \lhd A & \triangleq \exists_\sigma t.\ (D\ t \lhd A).
\end{array}$$

$$\triangleright A \triangleq \exists D(\mathit{atomic}\ A \wedge \mathit{prog}\ D \wedge D \lhd A).$$

Fig. 2. Interpreter for object-level specifications.

$$\begin{array}{rl}
\mathit{atomic}\ (M \Downarrow V) & \triangleq \top. \\
\mathit{prog}\ (\hat{\forall} B \hat{\forall} M \hat{\forall} N \hat{\forall} V[B \Downarrow \mathit{true} \mathbin{\&} M \Downarrow V \Rightarrow (\mathit{if}\ B\ M\ N) \Downarrow V]) & \triangleq \top. \\
\mathit{prog}\ (\hat{\forall} B \hat{\forall} M \hat{\forall} N \hat{\forall} V[B \Downarrow \mathit{false} \mathbin{\&} N \Downarrow V \Rightarrow (\mathit{if}\ B\ M\ N) \Downarrow V]) & \triangleq \top.
\end{array}$$

Fig. 3. Examples of object-level Horn clauses.

this interpreter here is so that we may elaborate it in the next section to help capture reasoning about generic judgments.

3.3 Generic Judgments as Atomic Meta-level Judgments

When using HOAS or λ-tree syntax representations, inference rules of the form $\dfrac{\hat{\forall} x.Gx}{A}$ are often encountered. If one were to capture this in the interpreter described in Figure 2, there would need to be a way to interpret universally quantified goals. One obvious such interpretation is via the clause

$$\triangleright(\hat{\underset{\sigma}{\forall}} x.G\ x) \triangleq \forall_\sigma x[\triangleright G\ x], \tag{1}$$

That is, the object-level universal quantifier would be interpreted using the meta-level universal quantifier. While this is a common approach to dealing with object-level universal quantification, this encoding causes some problems when attempting to reason about logic specifications containing generic judgments.

For example, consider proving the query $\forall y_1 \forall y_2[q\ \langle y_1, t_1\rangle\ \langle y_2, t_2\rangle\ \langle y_2, t_3\rangle]$, where $\langle\cdot,\cdot\rangle$ is used to form pairs, from the three clauses $q\ X\ X\ Y$, $q\ X\ Y\ X$ and $q\ Y\ X\ X$. This query succeeds only if t_2 and t_3 are equal. In particular, we would like to prove the sequent

$$\triangleright(\hat{\forall} y_1 \hat{\forall} y_2[q\ \langle y_1, t_1\rangle\ \langle y_2, t_2\rangle\ \langle y_2, t_3\rangle]) \longrightarrow t_2 = t_3,$$

where t_1, t_2, and t_3 are eigenvariables and with a definition that consists of the clause (1), those clauses in Figure 2, and the following clauses:

$$\begin{array}{c}
X = X \triangleq \top \\
\mathit{atomic}\ (q\ X\ Y\ Z) \triangleq \top \\
\mathit{prog}\ (\hat{\forall} X \hat{\forall} Y\ q\ X\ X\ Y) \triangleq \top \\
\mathit{prog}\ (\hat{\forall} X \hat{\forall} Y\ q\ X\ Y\ X) \triangleq \top \\
\mathit{prog}\ (\hat{\forall} X \hat{\forall} Y\ q\ Y\ X\ X) \triangleq \top
\end{array}$$

Using these definitional clauses, this sequent reduces to

$$\triangleright(q\ \langle s_1, t_1\rangle\ \langle s_2, t_2\rangle\ \langle s_2, t_3\rangle) \longrightarrow t_2 = t_3,$$

for some terms s_1 and s_2. This latter sequent is provable only if s_1 and s_2 are chosen to be two non-unifiable terms. This style proof is quite unnatural and it also depends on the fact that the underlying type that is quantified in $\forall y_1 \forall y_2$ contains at least two distinct elements.

Additionally, if we consider a broader class of computational systems, other than conventional logical systems, a similar issue appears when we try to encode their term-level abstractions at the meta-level. For systems where names can be given scope and can be compared (the π-calculus, for example), interpreting object-level abstraction via universal quantifier may not be appropriate in certain cases. In the case of π-calculus, consider encoding the one-step transition semantics for the restriction operator:

$$(x)Px \xrightarrow{A} (x)P'x \triangleq \forall x.Px \xrightarrow{A} P'x.$$

If the atomic judgment $(x)Px \xrightarrow{A} (x)P'x$ is provable, then the meta-level atomic formula $Pt \xrightarrow{A} P't$ is provable for all names t. This is certainly not true in the present of match or mismatch operator in P. (We return to the π-calculus in more detail in Section 4.)

For these reasons, the uniform analysis of an object-level universal quantifier with a universal quantifier in (1) should be judged inappropriate. We now look for a different approach for encoding object logics.

A universal formula can be proved by an inference rule such as

$$\frac{\Gamma, c : \sigma \vdash Pc}{\Gamma \vdash \hat{\forall}_\sigma\, x.Px}\ ,$$

where P is a variable of higher type, Γ is a set of distinct typed variables, and c is a variable that is not free in the Γ nor in $\hat{\forall}_\sigma\, x.Px$. When we map the judgment $\Gamma \vdash \hat{\forall}_\sigma\, x.Px$ into the meta-logic, the entire judgment will be seen as an atomic meta-level formula: that is, we do not encode just $\triangleright(\hat{\forall}_\sigma\, x.Px)$ but rather the entire judgment $\triangleright(\Gamma \vdash \hat{\forall}_\sigma\, x.Px)$.

We consider two encodings of the judgment $x_1, \ldots, x_n \vdash (Px_1 \ldots x_n)$, where the variables on the left are all distinct. The first encoding introduces a "local" binders using a family of constants, say, loc_σ of type $(\sigma \to obj) \to obj$. The above expression would be something of the form

$$loc_{\sigma_1} \lambda x_1 \ldots loc_{\sigma_n} \lambda x_n.\ Px_1 \ldots x_n,$$

where, for $i = 1, \ldots, n$, σ_i is the type of the variable x_i. While this encoding is natural, it hides the top-level structure of $Px_1 \ldots x_n$ under a prefix of varying length. Unification and matching, which are central to the functioning of the definition introduction rules, does not directly access that top-level

$$\begin{aligned}
\rhd_l(\hat{\top}) &\triangleq \top. \\
\rhd_l((G\ l)\,\&\,(G'\ l)) &\triangleq \rhd_l(Gl) \wedge \rhd_l(G'l). \\
\rhd_l(\hat{\exists}_\sigma\, w.(G\ l\ w)) &\triangleq \exists_{evs\to\sigma} t.\, \rhd_l(G\ l\ (t\ l)). \\
\rhd_l(\hat{\forall}_\sigma\, w.(G\ l\ w)) &\triangleq \rhd_l(G(\hat{\rho} l)(\rho_\sigma l)). \\
\rhd_l(Al) &\triangleq \exists D.(atomic\ A \wedge prog D \wedge (Dl) \lhd_l (Al)). \\
(Al) \lhd_l (Al) &\triangleq atomic\ A. \\
((G\ l) \Rightarrow (D\ l)) \lhd_l (Al) &\triangleq (Dl) \lhd_l (Al) \wedge \rhd_l(Gl). \\
(\hat{\forall}_\sigma\, w.(D\ l\ w)) \lhd_l (Al) &\triangleq \exists_{evs\to\sigma} t.(D\ l\ (t\ l) \lhd_l (Al)).
\end{aligned}$$

Fig. 4. An interpreter for simple generic judgments.

structure. The second alternative employs a coding technique used by McDowell [McD97,MM02]. Here, one abstraction, say for a variable l of type *evs* (eigenvariables), is always written over the judgment and is used to denote the list of distinct variables $x_1, \ldots, x_n$. Individual variables are then accessed via the projections ρ_σ of type $evs \to \sigma$ and $\hat{\rho}$ of type $evs \to evs$. For example, the judgment $x : a, y : b, z : c \vdash Pxyz$ could be encoded as either the expression $loc_a \lambda x loc_b \lambda y loc_c \lambda z.Pxyz$, or as $\lambda l(P(\rho_a l)(\rho_b(\hat{\rho} l))(\rho_c(\hat{\rho}(\hat{\rho} l))))$. In this second, preferred encoding, the abstraction l denotes a list of variables, the first variable being of type a, the second being of type b, and the third of type c.

3.4 An Interpreter for Generic Judgments

To illustrate this style encoding of generic judgments, consider the interpreter displayed in Figure 4. This interpreter specifies provability for object-level Horn clauses in which the body of clauses may have occurrences of the universal quantifiers (as well as true, conjunction, and the existential quantifier) and generalizes the previous interpreter (Figure 2). Here, the three meta-level predicates *atomic* $\cdot$, *prog* $\cdot$, and $\rhd \cdot$ all have the type $(evs \to obj) \to o$ while $\cdot \lhd \cdot$ has the type $(evs \to obj) \to (evs \to obj) \to o$. The *evs* abstractions in these predicates are abbreviated as the subscript l next to the predicates, so the expression $\rhd((\lambda l.G\ l)\,\&\,(\lambda l.G'\ l))$ is written simply as $\rhd_l((G\ l)\,\&\,(G'\ l))$. Notice that the technique of replacing the abstraction $\lambda l.\hat{\forall}_\sigma\, \lambda w.(G\ l\ w))$ with $\lambda l.G(\hat{\rho} l)(\rho l))$ corresponds to replacing the judgment $x_1, \ldots, x_n \vdash \forall y(Pyx_1 \ldots x_n)$ with $x_1, \ldots, x_n, x_{n+1} \vdash (Px_1 \ldots x_n x_{n+1})$.

Notice that proof search using this interpreter will generate more than L_λ unification problems (via the left and right definition rules) for two reasons. First, the definition clause for interpreting $(\lambda l.\hat{\forall}_\sigma\, w.(G\ l\ w))$ contains the expression $(\lambda l.G(\hat{\rho} l)(\rho_\sigma l))$ and the subterms $(\hat{\rho} l)$ and $(\rho_\sigma l)$ are not distinct bound variables. However, the technical definition of L_λ can be extended to allow for this style of encoding of object-level variables, while still preserving the decidability and the mgu property of the unification problems [Tiu02]. A second reason that this interpreter is not in L_λ is the definition clause for backchaining over $(\lambda l.\hat{\forall}_\sigma\, w.(D\ l\ w))$ since this clause contains the expression $(\lambda l.D\ l\ (t\ l))$, which requires applying an abstraction to a general (external) term t. Such a specification can be made into an L_λ specification by encoding object-level substitution as an explicit judgment [Mil91b]. The fact that this specification is not in L_λ simply means that when we apply the left-introduction rule for definitions, unification may not produce a most general unifier.

See [MM02] for the correctness proofs of this encoding of generic judgments. Other judgments besides generic judgments can be encoded similarly. For example, in [McD97,MM02], hypothetical as well as linear logic judgments were encoded along these lines. The main objective in those papers is to encode an object-level sequent as atomic judgments in a meta-logic.

4 The π-Calculus

To illustrate the use of this style of representation of universal judgments, we turn, as many others have done [MP99,HMS01,Des00,RHB01], to consider encoding the π-calculus. In particular, we follow the encoding in [MP99] for the syntax and one-step operational semantics.

Following the presentation of the π-calculus given in [MPW92], we shall require three primitive syntactic categories: *name* for channels, *proc* for processes, and *action* for actions. The output prefix is the constructor *out* of type $name \rightarrow name \rightarrow proc \rightarrow proc$ and the input prefix is the constructor *in* of type $name \rightarrow (name \rightarrow proc) \rightarrow proc$: the π-calculus expressions $\bar{x}y.P$ and $x(y).P$ are represented as $(out\ x\ y\ P)$ and $(in\ x\ \lambda y.P)$, respectively. We use $|$ and $+$, both of type $proc \rightarrow proc \rightarrow proc$ and written as infix, to denote parallel composition and summation, and ν of type $(name \rightarrow proc) \rightarrow proc$ to denote restriction. The π-calculus expression $(x)P$ will be encoded as $\nu\lambda n.P$, which itself is abbreviated as simply $\nu x.P$. The match operator, $[\cdot = \cdot]\cdot$ is of type $name \rightarrow name \rightarrow proc \rightarrow proc$. When τ is written as a prefix, it has type $proc \rightarrow proc$. When τ is written as an action, it has type *action*. The symbols $\downarrow$ and $\uparrow$, both of type $name \rightarrow name \rightarrow action$, denote the input and output actions, respectively, on a named channel with a named value.

We shall deal with only *finite* π-calculus expression, that is, expressions without ! or defined constants. Extending this work to infinite process expressions can be done using induction, as outlined in [MMP01] or by adding an explicit co-induction proof rule dual to the induction rule. Fortunately, the finite expressions are rich enough to illustrate the issues regarding syntax and abstractions that are the focus of this paper.

Proposition 1. *Let P be a finite π-calculus expression using the syntax of [MPW92]. If the free names of P are admitted as constants of type name then P corresponds uniquely to a $\beta\eta$-equivalence class of terms of type proc.*

We first consider the encoding of one-step transition semantics of π-calculus. As we did with the encoding of object logics, we explicitly encode the "signatures" of π-calculus expressions (i.e., the names) as abstractions of type *evs*. The encoding uses two predicates: $\cdot \overset{\cdot}{\longrightarrow} \cdot$ of type $(evs \rightarrow proc) \rightarrow (evs \rightarrow action) \rightarrow$

$$\frac{}{\tau.P \xrightarrow{\tau} P}\,\tau \qquad \frac{P \xrightarrow{A} Q}{[x=x]P \xrightarrow{A} Q}\,\text{match} \qquad \frac{P \xrightharpoonup{A} Q}{[x=x]P \xrightharpoonup{A} Q}\,\text{match}$$

$$\frac{P \xrightarrow{A} R}{P+Q \xrightarrow{A} R}\,\text{sum} \qquad \frac{Q \xrightarrow{A} R}{P+Q \xrightarrow{A} R}\,\text{sum} \qquad \frac{P \xrightharpoonup{A} R}{P+Q \xrightharpoonup{A} R}\,\text{sum} \qquad \frac{Q \xrightharpoonup{A} R}{P+Q \xrightharpoonup{A} R}\,\text{sum}$$

$$\frac{P \xrightarrow{A} P'}{P|Q \xrightarrow{A} P'|Q}\,\text{par} \qquad \frac{Q \xrightarrow{A} Q'}{P|Q \xrightarrow{A} P|Q'}\,\text{par}$$

$$\frac{P \xrightharpoonup{A} M}{P|Q \xrightharpoonup{A} \lambda n(Mn|Q)}\,\text{par} \qquad \frac{Q \xrightharpoonup{A} N}{P|Q \xrightharpoonup{A} \lambda n(P|Nn)}\,\text{par}$$

$$\frac{\forall n(Pn \xrightarrow{A} P'n)}{\nu n.Pn \xrightarrow{A} \nu n.P'n}\,\text{res} \qquad \frac{\forall n(Pn \xrightharpoonup{A} P'n)}{\nu n.Pn \xrightharpoonup{A} \lambda m\ \nu n.(P'nm)}\,\text{res}$$

$$\frac{}{\mathit{out}\ x\ y\ P \xrightarrow{\uparrow xy} P}\ \text{output} \qquad \frac{}{\mathit{in}\ x\ M \xrightharpoonup{\downarrow x} M}\ \text{input}$$

$$\frac{\forall y(My \xrightarrow{\uparrow xy} M'y)}{\nu y.My \xrightharpoonup{\uparrow x} M'}\ \text{open}$$

$$\frac{P \xrightharpoonup{\downarrow x} M \qquad Q \xrightharpoonup{\uparrow x} N}{P|Q \xrightarrow{\tau} \nu n.(Mn|Nn)}\ \text{close} \qquad \frac{P \xrightharpoonup{\uparrow x} M \qquad Q \xrightharpoonup{\downarrow x} N}{P|Q \xrightarrow{\tau} \nu n.(Mn|Nn)}\ \text{close}$$

Fig. 5. The core π-calculus in λ-tree syntax.

$(\mathit{evs} \to \mathit{proc}) \to o$; and $\cdot \overset{\cdot}{\rightharpoonup} \cdot$ of type $(\mathit{evs} \to \mathit{proc}) \to (\mathit{evs} \to \mathit{name} \to \mathit{action}) \to (\mathit{evs} \to \mathit{name} \to \mathit{proc}) \to o$. The first of these predicates encodes transitions involving free values and the second encodes transitions involving bound values.

Figures 5, 6 and 7 present the transition semantics as Horn specification. We omit the *evs* abstraction for simplicity of presentation. We shall see later how to translate this specification into Horn definition clauses. Figure 5 specifies the one step transition system for the "core" π-calculus. Figure 6 provides the increment to the core rules to get the late transition system, and Figure 7 gives the increment to the core to get the early transition system. Note that all the rules in the core system belong to the L_λ subset of logic specifications: that is, abstractions are applied to only abstracted variables (either bound by a λ-abstraction or bound by a universally quantifier in the premise of the rule). Furthermore, note that each of the increments for the late and early systems involve at least one clause that is not in L_λ.

Sangiorgi has also motivated the same core system of rules [San96] and named them π_I: this subset of the π-calculus allows for "internal" mobility of names, that is, local (restricted) names only being passed to abstractions (via β_0 re-

$$\frac{P \xrightharpoonup{\downarrow x} M \qquad Q \xrightarrow{\uparrow xy} Q'}{P|Q \xrightarrow{\tau} (My)|Q'}\ \text{L-com} \qquad \frac{P \xrightarrow{\uparrow xy} P' \qquad Q \xrightharpoonup{\downarrow x} N}{P|Q \xrightarrow{\tau} P'|(Ny)}\ \text{L-com}$$

Fig. 6. The additional rules for late π-calculus.

$$\frac{}{in\ x\ M \xrightarrow{\downarrow xy} My}\ \text{E-input}$$

$$\frac{P \xrightarrow{\uparrow xy} P' \qquad Q \xrightarrow{\downarrow xy} Q'}{P|Q \xrightarrow{\tau} P'|Q'}\ \text{E-com} \qquad \frac{Q \xrightarrow{\downarrow xy} Q' \qquad P \xrightarrow{\uparrow xy} P'}{P|Q \xrightarrow{\tau} P'|Q'}\ \text{E-com}$$

Fig. 7. The additional rules for early π-calculus.

ductions), and disallows "external" mobility (via the more general β reduction rules).

One advantage of this style of specification over the traditional one [MPW92] is the absence of complicated side-conditions on variables: they are handled directly by the treatment of abstractions by logic.

Universally quantified premises, of which there are three in Figure 5, is one of the logical features that aids in encoding abstractions. When the inference rules in these figures are written as formulas, cut-free provability of the judgments $P \xrightarrow{A} Q$ and $P \xrightharpoonup{A} Q$ corresponds to one-step transitions for the π-calculus. While the right-introduction rule for the universal quantifiers that appears in these premises is correct, the corresponding left-introduction rule (which does not appear in cut-free proofs of just one-step transitions) for universal quantifiers is not always appropriate. It is the left-hand introduction rules for universal quantifiers that allow for arbitrary substitutions for abstractions, and, as suggested in Section 3.3, this property is certainly undesirable if we wish to extend our specification of the π-calculus, for example, to include the mismatch operator. We can also encode these Horn clauses as *prog* clauses in an object-logic interpreter like the one we defined in Section 3.4. This style of encoding inherits similar substitution properties of the object-logic.

One way to address this problem of the universal quantifier is to translate these inference rules into Horn definitions using the technique described in the previous section. We first need to make explicit the object-level eigenvariables by writing down the *evs* abstractions. Then, it is almost straightforward to rewrite these inference rule into Horn definitions, except for the cases involving the restriction operator which we will show here for the rules res and open.

$$\begin{array}{ll} \text{res:} & \nu n.Pln \xrightarrow{Al}_l \nu n.P'ln \triangleq P(\hat{\rho}l)(\rho l) \xrightarrow{A(\hat{\rho}l)}_l P'(\hat{\rho}l)(\rho l) \\ \text{open:} & \nu y.Mly \xrightharpoonup{\uparrow(xl)}_l M'l \triangleq M(\hat{\rho}l)(\rho l) \xrightharpoonup{\uparrow(x(\hat{\rho}l))(\rho l)}_l M'(\rho l) \end{array}$$

Here, the expression $\lambda l.Pl \xrightarrow{\lambda l.Al}_l \lambda l.P'l$ is abbreviated as $Pl \xrightarrow{Al}_l P'l$.

$$\begin{aligned}
sim_l\ (Pl)\ (Ql) \triangleq\ & \forall A \forall P' [(Pl \xrightarrow{Al}_l P'l) \supset \exists Q'\,(Ql \xrightarrow{Al}_l Q'l) \wedge \\
& \qquad\qquad\qquad\qquad\qquad sim_l\ (P'l)\ (Q'l)] \wedge \\
& \forall X \forall P' [(Pl \xrightarrow{\downarrow(Xl)}_l P'l) \supset \exists Q'\,(Ql \xrightarrow{\downarrow(Xl)}_l Q'l) \wedge \\
& \qquad\qquad\qquad\qquad\qquad \forall w\ sim_l\ (P'l(wl))\ (Q'l(wl))] \wedge \\
& \forall X \forall P' [(Pl \xrightarrow{\uparrow(Xl)}_l P'l) \supset \exists Q'\,(Ql \xrightarrow{\uparrow(Xl)}_l Q'l) \wedge \\
& \qquad\qquad\qquad\qquad\qquad sim_l\ (P'(\hat{\rho}l)(\rho l))\ (Q'(\hat{\rho}l)(\rho l))]
\end{aligned}$$

Fig. 8. Definition clause for simulation of π-calculus.

Proposition 2. *Let $P \xrightarrow{A} Q$ be provable in late (resp., early) transition system of [MPW92]. If A is either τ or $\downarrow xy$ or $\uparrow xy$ then $Pl \xrightarrow{Al}_l Ql$ is provable from the definition clauses encoding late (resp., early) transitions. (Here, P, A, and Q are all translated to the corresponding logic-level term.) If A is $x(y)$ then $Pl \xrightarrow{\downarrow(xl)}_l Rl$ and if A is $\bar{x}(y)$ then $Pl \xrightarrow{\uparrow(xl)}_l Rl$. Here, R is the representation of the λ-abstraction of y over Q.*

Since the types of P and P' are different in the expression $P \xrightarrow{A} P'$, we cannot immediately form the transitive closure of this relationship to give a notion of a sequence of transitions. It is necessary to lower the type of P' first by applying it to a term of type *name*. How this is done, depends on what we are trying to model. To illustrate two such choices, we now consider encoding simulation.

For simplicity, we shall consider only simulation and not bisimulation: extending to bisimulation is not difficult (see Figure 1) but does introduce several more cases and make our examples more difficult to read. Figure 8 presents a definition clause for simulation. Here, the predicate *sim* is of type $(evs \to proc) \to (evs \to proc) \to o$ and again, we abbreviate the expression $sim\ (\lambda l.Pl)\ (\lambda l.Ql)$ as $sim_l(Pl)(Ql)$. Here, X has type $evs \to name$, P has type $evs \to proc$, and P' has two different types, $evs \to proc$ and $evs \to name \to proc$. Since the only occurrence of ρ_σ is such that σ is *name*, we shall drop the subscript on ρ. Notice also that for this set of clauses to be successfully stratified, the level of *sim* must be strictly greater than the level of all the other predicates of the one-step transitions (which can all be equal).

The first conjunct in the body of the clause in Figure 8 deals with the case where a process makes either a τ step or a free input or output action. In these cases, the variable A would be bound to either $\lambda l.\tau$ (in the first case) or $\lambda l. \downarrow (N\ l)(M\ l)$ or $\lambda l. \uparrow (N\ l)(M\ l)$, in which cases, N and M would be of the form $\lambda l.\rho(\hat{\rho}^i l)$ for some non-negative integer i.

The last two cases correspond to when a bounded input or output action is done. In the case of the bounded input (the second conjunct), a universal quantifier, $\forall w$, is used to instantiate the abstractions (P' and Q'), whereas in the bounded output case (the third conjunct), a term-level abstraction is emulated:

such an internal abstraction is immediately replaced by using a new variable in the context, via the use of ρ and $\hat{\rho}$. This one definition clause thus illustrates an important distinction between term-level and proof-level abstractions: in particular, they reflect the different behaviors of name-binding constructs in input prefix and restriction operator of π-calculus.

5 Related and Future Work

Of course, the value of this approach to encoding object-logics and the π-calculus comes, in part, from the ability to automate proofs using such definitions. Jeremie Wajs and Miller have been developing a tactic-style theorem prover for a logic with induction and definitions [Waj02]. This system, called Iris, is written entirely in Nadathur's Teyjus implementation [NM99] of λProlog and appears to be the first theorem proving system to be written entirely using higher-order abstract syntax (parser, printer, top-level, tactics, tacticals, etc). Example proofs that we have done by hand come out as expected: they are rather natural and immediate, although the encoding of object-level signature as a single abstraction makes expressions rather awkward to read. Fortunately, simple printing and parsing conventions can improve readability greatly. Given that the interpreter in Figure 4 is based on Horn clauses definitions, it is possible to employ well known induction principles for Horn clauses to help prove properties of the object logics/systems.

There are various other calculi, such as the join and ambient calculi, in which names and name restriction are prominent and we plan to test this style of encoding with them. Generic judgments have been used to model names for references and exceptions [Mil96,Chi95] and it would be interesting to see if this style of encoding can help in reasoning about programming language semantics containing such features. Comparing this particular encoding of the π-calculus with those of others, for example, [HMS01,Des00,RHB01], should be done in some detail.

Finally, the approach to encoding syntax and operational semantics used here is strongly motivated by proof theoretic considerations. There has been much work lately on using a more model-theoretic or categorical-theoretic approaches for such syntactic representations, see for example [FPT99,GP99,Hof99]. Comparing those two approaches should be illuminating.

Acknowledgments

Catuscia Palamidessi has made a number of comments on a draft of this paper. Anonymous reviewers for MERLIN 2001 also made several corrections to the presentation of this paper. This work is funded in part by NSF grants CCR-9912387, CCR-9803971, INT-9815645, and INT-9815731.

References

Chi95. Jawahar Chirimar. *Proof Theoretic Approach to Specification Languages.* PhD thesis, University of Pennsylvania, February 1995.

Des00. Jolle Despeyroux. A higher-order specification of the π-calculus. In *Proc. of the IFIP International Conference on Theoretical Computer Science, IFIP TCS'2000, Sendai, Japan, August 17-19, 2000.*, August 2000.

FPT99. M. P. Fiore, G. D. Plotkin, and D. Turi. Abstract syntax and variable binding. In *14th Annual Symposium on Logic in Computer Science*, pages 193–202. IEEE Computer Society Press, 1999.

Gir92. Jean-Yves Girard. A fixpoint theorem in linear logic. Email to the linear@cs.stanford.edu mailing list, February 1992.

GP99. M. J. Gabbay and A. M. Pitts. A new approach to abstract syntax involving binders. In *14th Annual Symposium on Logic in Computer Science*, pages 214–224. IEEE Computer Society Press, 1999.

HMS01. Furio Honsell, Marino Miculan, and Ivan Scagnetto. Pi-calculus in (co)inductive type theory. *TCS*, 253(2):239–285, 2001.

Hof99. M. Hofmann. Semantical analysis of higher-order abstract syntax. In *14th Annual Symposium on Logic in Computer Science*, pages 204–213. IEEE Computer Society Press, 1999.

HSH91. Lars Hallnäs and Peter Schroeder-Heister. A proof-theoretic approach to logic programming. ii. Programs as definitions. *Journal of Logic and Computation*, 1(5):635–660, October 1991.

Hue75. Gérard Huet. A unification algorithm for typed λ-calculus. *TCS*, 1:27–57, 1975.

Kah87. Gilles Kahn. Natural semantics. In *Proc. of the Symposium on Theoretical Aspects of Computer Science*, volume 247 of *LNCS*, pages 22–39. March 1987.

McD97. Raymond McDowell. *Reasoning in a Logic with Definitions and Induction.* PhD thesis, University of Pennsylvania, December 1997.

Mil90. Dale Miller. Abstractions in logic programming. In Piergiorgio Odifreddi, editor, *Logic and Computer Science*, pages 329–359. Academic Press, 1990.

Mil91a. Dale Miller. A logic programming language with lambda-abstraction, function variables, and simple unification. *Journal of Logic and Computation*, 1(4):497–536, 1991.

Mil91b. Dale Miller. Unification of simply typed lambda-terms as logic programming. In *Eighth International Logic Programming Conference*, pages 255–269, Paris, France, June 1991. MIT Press.

Mil96. Dale Miller. Forum: A multiple-conclusion specification language. *TCS*, 165(1):201–232, September 1996.

Mil00. Dale Miller. Abstract syntax for variable binders: An overview. In John Lloyd and et. al., editors, *Computational Logic - CL 2000*, number 1861 in LNAI, pages 239–253. Springer, 2000.

Mil01. Dale Miller. Encoding generic judgments: Preliminary results. In R.L. Crole S.J. Ambler and A. Momigliano, editors, *ENTCS*, volume 58. Elsevier, 2001. Proceedings of the MERLIN 2001 Workshop, Siena.

MM97. Raymond McDowell and Dale Miller. A logic for reasoning with higher-order abstract syntax. In Glynn Winskel, editor, *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science*, pages 434–445, Warsaw, Poland, July 1997. IEEE Computer Society Press.

MM00. Raymond McDowell and Dale Miller. Cut-elimination for a logic with definitions and induction. *TCS*, 232:91–119, 2000.

MM02. Raymond McDowell and Dale Miller. Reasoning with higher-order abstract syntax in a logical framework. *ACM Transactions on Computational Logic*, 3(1):80–136, January 2002.

MMP01. Raymond McDowell, Dale Miller, and Catuscia Palamidessi. Encoding transition systems in sequent calculus. *TCS*, 197(1-2), 2001. To appear.

MN85. Dale Miller and Gopalan Nadathur. A computational logic approach to syntax and semantics. Presented at the Tenth Symposium of the Mathematical Foundations of Computer Science, IBM Japan, May 1985.

MN86. Dale Miller and Gopalan Nadathur. Some uses of higher-order logic in computational linguistics. In *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*, pages 247–255. Association for Computational Linguistics, Morristown, New Jersey, 1986.

MN87. Dale Miller and Gopalan Nadathur. A logic programming approach to manipulating formulas and programs. In Seif Haridi, editor, *IEEE Symposium on Logic Programming*, pages 379–388, San Francisco, September 1987.

MP99. Dale Miller and Catuscia Palamidessi. Foundational aspects of syntax. In Pierpaolo Degano, Roberto Gorrieri, Alberto Marchetti-Spaccamela, and Peter Wegner, editors, *ACM Computing Surveys Symposium on Theoretical Computer Science: A Perspective*, volume 31. ACM, Sep 1999.

MPW92. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, Part II. *Information and Computation*, pages 41–77, 1992.

Nip91. Tobias Nipkow. Higher-order critical pairs. In G. Kahn, editor, *LICS91*, pages 342–349. IEEE, July 1991.

NM99. Gopalan Nadathur and Dustin J. Mitchell. System description: Teyjus—a compiler and abstract machine based implementation of Lambda Prolog. In H. Ganzinger, ed., *CADE16*, pages 287–291, Trento, Italy, July 1999. LNCS.

Pau89. Lawrence C. Paulson. The foundation of a generic theorem prover. *Journal of Automated Reasoning*, 5:363–397, September 1989.

PE88. Frank Pfenning and Conal Elliott. Higher-order abstract syntax. In *Proceedings of the ACM-SIGPLAN Conference on Programming Language Design and Implementation*, pages 199–208. ACM Press, June 1988.

Plo81. G. Plotkin. A structural approach to operational semantics. DAIMI FN-19, Aarhus University, Aarhus, Denmark, September 1981.

RHB01. C. Röckl, D. Hirschkoff, and S. Berghofer. Higher-order abstract syntax with induction in Isabelle/HOL: Formalizing the pi-calculus and mechanizing the theory of contexts. In *Proceedings of FOSSACS'01*, LNCS, 2001.

San96. Davide Sangiorgi. π-calculus, internal mobility and agent-passing calculi. *TCS*, 167(2):235–274, 1996.

SH92. Peter Schroeder-Heister. Cut-elimination in logics with definitional reflection. In D. Pearce and H. Wansing, editors, *Nonclassical Logics and Information Processing*, volume 619 of *LNCS*, pages 146–171. Springer, 1992.

SH93. Peter Schroeder-Heister. Rules of definitional reflection. In M. Vardi, editor, *Eighth Annual Symposium on Logic in Computer Science*, pages 222–232. IEEE Computer Society Press, IEEE, June 1993.

Tiu02. Alwen F. Tiu. An extension of L-lambda unification. Draft, available via `http://www.cse.psu.edu/~tiu/llambdaext.pdf`, September 2002.

Waj02. Jérémie D. Wajs. *Reasoning about Logic Programs Using Definitions and Induction*. PhD thesis, Pennsylvania State University, 2002.

Model Checking Algol-Like Languages Using Game Semantics

C.-H.L. Ong[1,2]

[1] Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, United Kingdom
lo@comlab.ox.ac.uk
[2] School of Computing, National University of Singapore

Abstract. We survey a recent development of Game Semantics in a new, algorithmic direction, with a view to applications in computer-assisted verification and program analysis.

Game Semantics has emerged as a powerful paradigm for giving accurate semantics to a variety of programming languages and logical systems. In the game reading, types (or specifications) are interpreted as games, and programs (or implementations) as strategies on games. Games are played between two players, known as P and O: P's perspective is that of the system (or program) being modelled, and O's is that of the environment (or program context). This simple and intuitive game-playing idea has given rise to *universal* models (i.e. every point of the model is the denotation of some term) for the untyped λ-calculus [12,11] and System F [9], and to *fully abstract* models (i.e. the induced theory of equality coincides with observational equivalence) for a spectrum of programming languages, ranging from the purely functional [2,10] to those with non-functional features such as control operators [13], locally-scoped [3] and higher-order references [1], erratic non-determinism [8] and probability [5].

Algorithmic Game Semantics: Some Recent Results

We survey a recent development of Game Semantics in a new, *algorithmic* direction, with a view to applications in computer-assisted verification and program analysis. Several features of Game Semantics make it very promising for such applications. Game Semantics provides a very *concrete* way of building *fully abstract* models. It has a clear operational content, while admitting *compositional methods* in the style of denotational semantics. The basic objects studied in Game Semantics are games, and strategies on games. Strategies can be seen as certain kinds of highly-constrained processes, hence they admit the same kind of automata-theoretic representations central to model checking (see e.g. [20]) and allied methods in computer-assisted verification. Moreover games and strategies naturally form themselves into rich mathematical structures which yield very accurate models of advanced high-level programming languages, as the various full abstraction results show.

The important first steps in this direction have been taken by Ghica and McCusker [6]; they consider Idealized Algol (IA) [16] (a compact language which elegantly combines state-based procedural and higher-order functional programming) and show that the 2nd-order finitary (i.e. recursion-free) fragment of the fully abstract game semantics of IA can be represented in a remarkably simple form by regular expressions. Precisely they prove that the complete plays of the game semantics of every term of the fragment form a regular language. Since complete plays characterize observational equivalence [3], their result yields a procedure for deciding observational equivalence for the fragment.

Further results about the decidability of observational equivalence for other fragments of IA have been obtained recently. By modelling state explicitly, the denotation of a 3rd-order finitary IA term can be shown to be *compactly innocent* i.e. generated by a finite view function in the sense of [10]. Any such function defines a deterministic pushdown automaton (DPDA) that recognizes exactly the complete plays of the game semantics of the term in question. Thus observational equivalence for the 3rd-order fragment is reduced to the decision problem

> DPDA Equivalence: Given two DPDAs, do they recognize the same language?

First posed in 1966 [7], DPDA Equivalence was only recently proved to be decidable by Sénizergues [18] who was awarded the 2002 EATCS/ACM Gödel Prize for his accomplishment (see also [19]). Hence observational equivalence for 3rd-order finitary IA is decidable [15].

Can the algorithmic representation be extended to 4th and higher orders? It turns out that 3rd order is the limit, and the best that we can do. By demonstrating that there are 4th-order terms whose complete plays correspond to runs of Turing-complete machine models, Murawski [14] has shown that observational equivalence is undecidable for the 4th-order fragment. Observational equivalence is also undecidable for the 2nd-order fragment augmented by a fixpoint operator (thus admiting recursively defined first-order functions) [15].

Distinctives of the Game Semantics Approach

This algorithmic representation of program meanings, which is compositional, provides a foundation for model-checking a wide range of behavioural properties of Algol-like programming languages. The promise of this approach is to transfer the methods of Model Checking (see e.g. [4]) based on automata-theoretic representations, which has been so successful in the analysis of hardware designs and communications and security protocols, to the much more *structured* setting of programming languages, in which data types and control flow are important.

A further benefit of the algorithmic approach is that by embodying game semantics in tools, and making it concrete and algorithmic, it should become more accessible and meaningful to practitioners. We see Game Semantics as having the potential to fill the role of a "Popular Formal Semantics" called for in an eloquent paper by Schmidt [17], which can help to bridge the gap between the semantics and programming language communities. Game Semantics has been successful in its own terms as a semantic theory; we aim to make it useful to and usable by a wider community.

In relation to the extensive current activity in model checking and computer assisted verification, our approach is distinctive, being founded on a highly-structured *compositional* semantic model. This means that we can directly apply our methods to *program phrases* (i.e. terms-in-context with free variables) in a high-level language with procedures, local variables and data types; moreover, the soundness of our methods is guaranteed by the properties of the semantic models on which they are based. By contrast, most current model checking applies to relatively "flat" unstructured situations, in which the system being analyzed is presented as a transition system or automaton. It seems fair to say that Software Model Checking is still in a comparatively primitive state, with tool development in several cases running well ahead of rigorous consideration of soundness of the methods being developed. Our aim is to build on the tools and methods which have been developed in the Verification community, while exploring the advantages offered by our semantics-based approach.

With colleagues at the Computing Laboratory, University of Oxford, work[1] is underway to investigate applications of Algorithmic Game Semantics to Software Model Checking. In addition to model-checking observational equivalence which is of basic importance, the same algorithmic representations of program meanings which are derived in this work can be put to use in verifying a wide range of program properties of IA and cognate programming languages, and in practice this is where the interesting applications are most likely to lie.

Acknowledgements

I gratefully acknowledge the work and influence of Samson Abramsky, Dan Ghica, Guy McCusker and Andrzej Murawski; they have all contributed much to the work reported here.

References

1. S. Abramsky, K. Honda, and G. McCusker. Fully abstract game semantics for general reference. In *Proceedings of IEEE Symposium on Logic in Computer Science, 1998*. Computer Society Press, 1998.
2. S. Abramsky, R. Jagadeesan, and P. Malacaria. Full abstraction for PCF. *Information and Computation*, 163, 2000.
3. S. Abramsky and G. McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P. W. O'Hearn and R. D. Tennent, editors, *Algol-like languages*. Birkhaüser, 1997.
4. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
5. V. Danos and R. Harmer. Probabilistic game semantics. In *Proc. IEEE Symposium on Logic in Computer Science, Santa Barbara, June, 2000*. Computer Science Society, 2000.

[1] For details of the Project, visit `http://web.comlab.ox.ac.uk/oucl/work/luke.ong`.

6. D. R. Ghica and G. McCusker. Reasoning about Idealized Algol using regular languages. In *Proceedings of 27th International Colloquium on Automata, Languages and Programming ICALP 2000*, pages 103–116. Springer-Verlag, 2000. LNCS Vol. 1853.
7. S. Ginsberg and S. Greibach. Deterministic context-free languages. *Information and Control*, pages 620–648, 1966.
8. R. Harmer and G. McCusker. A fully abstract game semantics for finite nondeterminism. In *Proceedings of Fourteenth Annual IEEE Symposium on Logic in Computer Science*. IEEE Computer Society Press, 1999.
9. D. H. D. Hughes. Games and definability for System F. In *Proceedings of 12th IEEE Symposium on Logic in Computer Science*. IEEE Computer Science Society, 1997.
10. J. M. E. Hyland and C.-H. L. Ong. On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model. *Information and Computation*, 163:285–408, 2000.
11. A. D. Ker, H. Nickau, and C.-H. L. Ong. A universal innocent game model for the Böhm tree lambda theory. In *Computer Science Logic: Proceedings of the 8th Annual Conference on the EACSL Madrid, Spain, September 1999*, pages 405 – 419. Springer-Verlag, 1999. LNCS Volume 1683.
12. A. D. Ker, H. Nickau, and C.-H. L. Ong. Innocent game models of untyped λ-calculus. *Theoretical Computer Science*, 272:247–292, 2002.
13. J. Laird. *A semantic analysis of control*. PhD thesis, University of Edinburgh, 1998.
14. A. Murawski. Finitary higher-order programming languages with first-order references are undecidable. Preprint, 2002.
15. C.-H. L. Ong. Observational equivalence of third-order Idealized Algol is decidable. In *Proceedings of IEEE Symposium on Logic in Computer Science, 22-25 July 200, Copenhagen Denmark*, pages 245–256. Computer Society Press, 2002.
16. J. C. Reynolds. The essence of Algol. In J. W. de Bakker and J. C. van Vliet, editors, *Algorithmic Languages*, pages 345–372. North Holland, 1978.
17. D. A. Schmidt. On the need for a popular formal semantics. *ACM SIGPLAN Notices*, 32:115–116, 1997.
18. G. Sénizergues. $L(A) = L(B)$? Decidability results from complete formal systems. *Theoretical Computer Science*, 251:1–166, 2001.
19. C. Stirling. Decidability of DPDA equivalence. *Theoretical Computer Science*, 255:1–31, 2001.
20. M. Y. Vardi. An automata-theoretic approach to linear temporal logic. In *Banff Higher Order Workshop*, pages 238–266. Springer-Verlag, 1995.

Modeling Software: From Theory to Practice

Margus Veanes

Microsoft Research, Redmond, WA, USA
margus@microsoft.com

Abstract. Modeling of software is hard. The most difficult task is to identify and remain faithful to the right levels of abstraction, to decide what should be in the model and what should be abstracted away. Abstract state machines (ASMs) give the modeler the maximal flexibility in dealing with this task. ASMs are able to simulate arbitrary computer systems on their natural level of abstraction. This is known as the *ASM thesis*; it has theoretical support and has been confirmed by numerous academic and industrial projects. In order to use ASMs for modeling industrial software, the group on Foundations of Software Engineering at Microsoft Research developed the *Abstract State Machine Language*. It provides a modern object-oriented, component-based specification environment complete with tools to test and enforce specifications.

1 Overview

To understand the need in software models, consider the question what the biggest challenges are in the software industry today. Undoubtedly, reducing the number and severity of bugs is one of them. There are several reasons why software is error-prone: ambiguous specifications, unforeseen feature interaction, premature coding, the need to change code in the last minute to incorporate new features, etc. By the time a software product is ready for shipping, its original specification may be and usually is way out of sync with the implementation. Specifications are typically written as natural language documents which adds to the gap between specifications and implementations and contributes to further problems during the maintenance phase of the software. There are many different kinds of error: coding defects, security violations, memory corruption (e.g. due to buffer overruns), design defects, etc. Typically, design defects are the worst bugs, since fixing them is so costly. In traditional engineering disciplines, such as architecture, mechanical engineering, etc., the role of models is relatively well understood. In software engineering, models are controversial. We think that models are indispensable for understanding the intended *behavior* of the system, for validating the spec and for enforcing the spec. In particular, models allow us to play out the system's user scenarios, to test feature interactions, and to examine various aspects of a larger system such as messaging, routing and security.

Software modeling approach needs to be *flexible* enough to adapt to a variety of modeling needs; the model needs to be *conceptually clear* so that validation against requirements is straightforward and so that it can be used as documentation; the model needs to be *mathematically precise* so it can be reasoned about; and – the last but definitely not the least – the model needs to be *executable* so that it can be experimented with and used for testing. The abstract state machines approach meets all these challenges, except that one needs a more pragmatic language to execute abstract state machines (ASMs) for real in an industrial environment; see section 4.

Abstract state machines (originally called evolving algebras) were introduced by Gurevich in a quest to capture the notion of *algorithm*. The *ASM Thesis*, first mentioned in [13], states that ASMs are *universal* in a strong sense with respect to *algorithms*, in particular *any algorithm can be step-for-step simulated by an ASM*. The notion of sequential algorithm is formalized in [14] by means of three postulates, and the sequential version of the ASM thesis is proved as a theorem there. Intuitively sequential algorithms are non-distributed algorithms with uniformly bounded parallelism, so that the number of actions performed in parallel is bounded independently of the state or the input. This has been extended to parallel synchronous algorithms in [6]. The work on proving the ASM thesis for distributed algorithms is in progress. By now there is a growing ASM community (see [1]) and numerous applications of ASMs in industry and academia [2,10]. The ability to simulate arbitrary algorithms on their natural level of abstraction has made ASMs attractive for high-level system design and analysis [7,8,9].

2 The Definition of Algorithm and Modeling Principles

Some computer science theory have brought a substantial advance in software technology. Automata theory, parsing theory, relational database theory come to mind. It is our opinion that ASM theory has the potential to make it to the big league. To illustrate the relevance of the ASM theory to modeling, we note that the postulates used to formalize the notion of algorithm have become important modeling principles. We restrict our attention to sequential algorithms.

- *Postulate 1* (*Sequential Time*). A sequential algorithm M is associated with a set of *states* $S(M)$, a subset $I(M)$ of *initial states*, and a *one-step transformation* map τ_M on states.
- *Postulate 2* (*Abstract State*). States of M are first-order structures over the same vocabulary; τ_M does not change the base set of the states; $S(M)$ and $I(M)$ are closed under isomorphisms, and if $f\colon A \cong B$ then $f\colon \tau_M(A) \cong \tau_M(B)$.

One begins building a model of a software system by considering the purpose of the model. Abstraction decisions are guided by Postulate 2. Those decisions are reflected in the choice of the vocabulary of the model, the universe and the interpretations of the vocabulary symbols. According to the second postulate, the nature of the basic elements themselves is immaterial.

The constraints that have to hold between various entities of the model give rise to the definitions of legal states and initial states. We have to define the transitions of the model, in other words, what is to be considered one step of the system that is being modeled at the given level of abstraction. This all corresponds to Postulate 1.

The model needs to be tuned up again and again to increase confidence that it is faithful to your intention. Typically, this leads to revisiting the modeling decisions that correspond to Postulates 1 and 2.

Sequential Algorithms. In the case of sequential algorithms, there is a uniform bound on the *amount of work* done during a single step of the algorithm. Given two states A and B with the same base set, let A - B denote the set of all tuples $(f, \boldsymbol{a}, f^A(\boldsymbol{a}))$ where f is a j-ary vocabulary function symbol and $\boldsymbol{a}$ is a j-tuple of basic elements such that $f^A(\boldsymbol{a}) \neq f^B(\boldsymbol{a})$. Intuitively, A - B is the part of A that is different from B.

- *Postulate 3* (*Bounded Exploration*). There is a *finite* set of terms T, such that, for all states A and B, if $T^A = T^B$ then $\tau_M(A) - A = \tau_M(B) - B$.

The third postulate gives precise meaning to the notion of bounded work. In terms of modeling, the corresponding requirement on the model is that the model program should be *explicit* about everything it does (without decreasing the abstraction level). If you do seven things at once, you mention explicitly each one of them. There should be no constructs that, depending on the state, may produce increasingly growing state modifications.

Non-sequential Algorithms. The bounded exploration postulate is too restrictive for most of our modeling purposes. We often model highly parallel and highly distributed systems involving agents or subsystems that are not bounded in this sense. Concerning the formalization of parallel algorithms and parallel ASMs, see [6].

3 Abstract State Machines

Here we give a quick description of ASMs. The interested reader may want to consult [6,12,14]. An ASM consists of an ASM *program* together with a collection of *states* and a subcollection of *initial states*.

The notion of ASM *state* is a slight variation of the notion of first-order structure. Truth values are elements of the base set. The value *undef* is the default value for basic functions. Formally a basic function f is total but intuitively it is partial; the intended domain of f consists of all tuples $\boldsymbol{a}$ with $f(\boldsymbol{a}) \neq undef$. Every state includes an infinite "naked" set called the *reserve*. The role of the reserve is to provide new elements for function domains.

A *location* of a state A is a pair $(f, \boldsymbol{a})$ where f is a j-ary dynamic function (or relation) symbol in the vocabulary of A and $\boldsymbol{a}$ is a j-tuple of elements of A. An *update* of A is a pair (l, b), where l is a location and b is an element of A. The *sequel* of *firing* a consistent (non-conflicting) finite set of updates $((f_i, \boldsymbol{a}_i), b_i)_{i\in I}$ on A is a state B that is exactly the same as A except that for all $i \in I$, $f_i(\boldsymbol{a}_i) = b_i$ in B.

Sequential ASM *rules* are defined inductively from *assignments*, *conditional statements* and *uniformly bounded parallel composition*. We do not define the formal syntax of rules here, but refer to the examples in the following section. When a program is fired on a given state A, a set of updates is produced that yields, in one atomic step, a sequel of A. Typically a program is fired repeatedly until no more changes occur in the state. The resulting sequence of states (possibly infinite) is called a *run*.

Sequential ASMs are ASMs whose programs are sequential rules.

- *Theorem* (*Formalization of the Sequential ASM Thesis*) [14]. For every sequential algorithm there is an abstract state machine with the same states, initial states and the same transition function.

The formalization of the sequential ASM thesis is examined in [17]. Parallel ASMs extend sequential ASMs. One can compose *comprehension expressions* and *parallel composition rules* that don't have to be uniformly bounded. Parallel ASMs are further extended to *nondeterministic ASMs* and to *distributed ASMs*; see [12].

4 AsmL

To use ASMs in an industrial environment, we need an industrial-strength language and toolkit. In addition to the general requirements mentioned above, there are more. The language must support *subtyping* that helps to define similarities, and *modularization* that helps to build manageable self-contained units. There must be a way to *enforce* conformance between the implementation and its specification. This calls for tools that are *compatible* and *interoperable*. The tools must be easily *portable* and *available*. Moreover, specification documents should live in Word or HTML instead of plain text, and the source should live in programming environments like Visual Studio instead of just being command line applications.

One such language has been (and is being) developed in Microsoft Research. It is called AsmL (ASM Language). The expressive power of AsmL is that of nondeterministic parallel ASMs with submachines. Moreover, AsmL has built-in support for data structures like (finite) sets and maps and supports partial updates to sets and maps. The concept of state background for ASMs is introduced in [5] and the theory of partial updates is studied in [15]. An extension of AsmL to incorporate distributed ASMs is in progress.

AsmL is a member of the .Net family of languages. It compiles to the intermediate language of the common language runtime and is interoperable with other .Net languages e.g. C# and VB. This interoperability is necessary for acceptance of AsmL by Microsoft product groups. It also allows one to build AsmL tools which require interoperability, such as on-line runtime verification tools [4]. The use of AsmL to specify behavioral interfaces of component is studied in [3].

In the following we explain briefly how the fundamental modeling concepts of ASMs are realized in AsmL. Some examples are given at the end of the section. For an in-depth introduction to AsmL and further examples of AsmL programs see the AsmL web site [2]. In particular, [11] is a good introduction to modeling with distributed ASMs.

Remark. There is a fair amount of freedom in AsmL regarding the way in which the modeled entities are represented. The reader should keep in mind that the particular choice of representation is often a matter of taste, readability and a way to control how the model is executed, and does not affect the underlying ASM semantics.

4.1 Types

Some ASM *universes* (i.e. unary relations) give rise to *types* in AsmL. Other universes are represented as (finite) sets; see the examples below. An AsmL program may first declare an abstract type *C* and later on concretize that type into a *class*, a *structure*, a finite *enumeration*, or a derived type.

```
type C
```

```
class C
```

AsmL has an expressive type system that allows one to define new types using (finite) sets, (finite partial) maps, (finite) sequences, tuples, etc., combined in arbitrary ways. For example, if *C* and *D* are types then the type of all maps from *C* to sets of elements of *D* is this.

```
Map of C to Set of D
```

4.2 Derived Functions

Derived functions play an important role in applications of ASMs. A derived function does not appear in the vocabulary; instead it is computed on the fly at a given state. In AsmL, derived functions are given by methods that return values. A derived function *f* from $C_0 \times C_1 \times ... \times C_n$ to *D* can be declared as a method.

```
f(x0 as C0, x1 as C1, ..., xn as Cn) as D
```

The definition of (how to compute) *f* may be given together with the declaration or introduced later on in the program. If C_0 is a class (or a structure) then *f* can be declared as a method of C_0.

```
class C0
  f(x1 as C1,...,xn as Cn) as D
```

4.3 Constants

A nullary function that does not change during the evolution of the state can be declared as a *constant*. Similarly, a unary static function *f* of type $C \to D$ where *C* is a class can be declared as a *constant field* of *C*.

```
z as Integer = 0
```

```
class C
  f as D
```

4.4 Variables

There are two kinds of variables in AsmL, *global variables* and *local variable fields* of classes. Semantically, fields of classes are *unary* functions. In the following, *b* is a propositional variable and *f* represents a unary dynamic function from *C* to integers.

```
var b as Boolean
```

```
class C
  var f as Integer
```

Dynamic functions of ASMs are represented by variables in AsmL. A dynamic function f of type $C_1 \times ... \times C_n \to D$ of any positive arity n can be represented as a *variable map* in AsmL.

```
var f as Map of (C1, ..., Cn) to D
```

4.5 Classes and Dynamic Universes

AsmL classes are special dynamic universes. Classes are initially empty. Let C and D be two dynamic universes such that C is a subset of D and let f be a dynamic function from C to integers.

```
class D

class C extends D

  var f as Integer
```

The following AsmL statement adds a new element c to C (and D) and initializes $f(c)$ to the value 0.

```
let c = new C(0)
```

Classes are special dynamic universes in that one cannot programmatically remove an element from a class. Moreover, classes cannot be quantified over. In order to keep track of elements of a class C, one can introduce (essentially an auxiliary dynamic universe represented by) a variable of type *set of C* that is initially empty.

```
var Cs as Set of C = {}

let c = new C(0)

add c to Cs
```

Sets can be quantified over like in the following rule where all the invocations of $R(x)$, one for every element x of the set s, are carried out simultaneously in one atomic step. Notice that the *forall* rule violates Postulate 3.

```
forall x in s

  R(x)
```

4.6 Examples

Here we give some sample specifications written in AsmL. The examples use some additional constructs not introduced above but their meaning should be clear from the context. In particular *choose* and *any* are used for expressing internal nondeterminism.

Topological sorting. This is a specification of topological sorting. Vertices are abstract objects. There is a nonempty set V of vertices.

```
class Vertex

V as Set of Vertex

require V <> {}
```

There is a binary relation *E* (the set of *edges*) on the vertices.

```
E as Set of (Vertex,Vertex)
```

Given an acyclic graph (*V*, *E*) the topological sorting algorithm builds a sequence *S* of vertices such that every edge leads forward. Initially the sequence is empty.

```
var S as Seq of Vertex = []
```

In one step, the algorithm picks a vertex *v* with no incoming edges from the vertices that are not in *S* yet and then moves the vertex *v* to *S*. (The built-in function *ToSet* converts a sequence to the set of elements in that sequence.)

```
Extend()

  let X = V - ToSet(S)

  if X <> {} then

    S := S + [(any v | v in X where

                   not(exists u in X where (u,v) in E))]
```

Notice that this specification does not impose any extraneous restrictions on the choice of *v*. The program runs until no more changes occur in the state. (If the given graph is cyclic then an exception will be generated: you cannot choose an element from the empty set.)

```
Topsort()

  step until fixpoint

    Extend()
```

Sorting. This example is the specification of any sequential *one-swap-at-a-time in-place* sorting algorithm. Examples of such algorithms are quicksort and insertion-sort. The sequence *A* of integers is to be sorted.

```
var A as Seq of Integer
```

In one step, two elements that are out of order are chosen nondeterministically and swapped. The specification does not say anything about the choice of the elements that are out of order. In other words, it abstracts from details of computing the indices *i* and *j*. The two assignments happen in parallel.

```
Swap()

  choose i in Indices(A), j in Indices(A)

         where i < j and A(i) > A(j)
      A(i) := A(j)
      A(j) := A(i)

  ifnone

    skip
```

The sorting algorithm terminates when no more swaps need to be made.

```
Sort()
  step until fixpoint
    Swap()
```

Simulation of Agents. Ideally one would like to have a distributed runtime environment for running distributed ASMs. The current version of AsmL doesn't have yet runtime support for true concurrency or simulation of true concurrency. Therefore one has to build functionality for simulating concurrent agents into the model. This is how it can be done.

Agents are introduced through a class *Agent*. To keep track of the currently active agents, a variable *Agents* of type *set of Agent* is updated each time an agent is created or discarded.

```
class Agent

var Agents as Set of Agent = {}
```

Each agent (more exactly, its state) evolves in sequential steps with each invocation of its *Program*. Each agent *a* has a *mailbox* of *messages* and a method *InsertMessage* that is used by other agents to send messages to *a*.

```
type Message

class Agent

  var mailbox as Set of Message = {}

  InsertMessage(m as Message)

    add m to mailbox

  Program()
```

The method *RunAgents* gives the operational semantics of a single step of the top-level system, in the definition of which *chooseSubset* selects nondeterministically a subset of the active agents. Thus, at each global step of the system, some, none or all of the active agents in the system may perform a step.

```
RunAgents()

  forall a in chooseSubset(Agents)

    Program(a)
```

5 From Modeling to Model-Based Testing

One of the huge benefits of having a rigorous model is the ability to reason about it in an objective manner. In particular, it is highly desirable to produce comprehensive and objective test suites from models and to use models as oracles in testing. The use of models in testing has therefore been receiving more and more attention by Microsoft product groups. Currently, the tools that are around require the models to be written as finite state machines (FSMs). However, direct FSM based modeling is often too restrictive and may lead to state space explosion. This has motivated us to study the problem of algorithmic generation of FSMs from AsmL specifications [16]. Runtime verification based on AsmL models is studied in [4].

6 Conclusions

Our group would like to improve software quality by making executable specifications an integral part of software development. We believe that ASMs are the right tool to accomplish the mission. This is supported by the ASM thesis and has been confirmed by numerous other industrial applications of ASMs. We developed an industrial-strength ASM Language and an AsmL toolkit for testing and enforcing ASM specifications [2]. We are gradually introducing AsmL to Microsoft product groups; it is being used by a number of product groups. There are still many obstacles to overcome. One of the most difficult obstacles is not technical but social: to change the attitude of software developers and teach them to appreciate and use executable specifications.

References

1. Abstract State Machines (ASMs), the academic Web site, http://www.eecs.umich.edu/gasm.
2. AsmL for Microsoft .Net, the ASM Language, http://research.microsoft.com/fse/asml
3. M. Barnett and W. Schulte. The ABCs of Specification: AsmL, Behavior, and Components, *Informatica*, 25(4), 2001.
4. M. Barnett and W. Schulte. Contracts, Components, and their Runtime Verification on the .NET Platform, Microsoft Research, Technical Report, MSR-TR-2002-38, 2002. To appear in *The Journal of Systems and Software*.
5. Blass and Y. Gurevich. Background, reserve, and Gandy machines, in *Proc. Computer Science Logic*, *Lecture Notes in Computer Science*, Vol. 1862, pages 1-17, Springer, 2000.
6. Blass and Y. Gurevich. Abstract State Machines Capture Parallel Algorithms. Microsoft Research, Technical Report, MSR-TR-2001-117, 2002. To appear in *ACM Transactions on Computational Logic*.
7. E. Börger. Why use evolving algebras for hardware and software engineering? *Lecture Notes in Computer Science*, Vol. 1012, pages 236-271, Springer, 1995.
8. E. Börger. High Level System Design and Analysis using Abstract State Machines. In D. Hutter, W. Stephan, P. Traverso, M. Ullman, eds., *Current Trends in Applied Formal Methods (FM-Trends 98)*. *Lecture Notes in Computer Science*, Vol. 1641, pp. 1-43, Springer, 1999.
9. E. Börger. The Origins and the Development of the ASM Method for High Level System Design and Analysis. *Journal of Universal Computer Science*, 2 (8): 2-74, Springer, 2002.
10. E. Börger and J. Huggins, Abstract state machines 1988-1998: Commented ASM bibliography, *Bulletin of EATCS*, Nr 64, pages 105-128, 1998.
11. U. Glässer, Y. Gurevich and M. Veanes. An Abstract Communication Model. Microsoft Research, Technical Report, MSR-TR-2002-55, 2002.
12. Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, Oxford University Press, pages 9-36, 1995.

13. Y. Gurevich, A new thesis, *American Mathematical Society Abstracts*, p. 317, 1985.
14. Y. Gurevich. Sequential Abstract State Machines Capture Sequential Algorithms, *ACM Transactions on Computational Logic*, vol. 1, no. 1, July 2000, pages 77-111.
15. Y. Gurevich and N. Tillmann. Partial Updates: Exploration. *Journal of Universal Computer Science*, 11 (7): 917-951, Springer, 2001.
16. W. Grieskamp, Y. Gurevich, W. Schulte and M. Veanes. Generating Finite State Machines from Abstract State Machines, *ISSTA 02, Software Engineering Notes* 27(4) 112-122, ACM, 2002.
17. W. Reisig. On Gurevich's Theorem for Sequential ASM, Informatik-Berichte Nr. 160, Humboldt-Universität zu Berlin, Juli 2002.

Local Normal Forms for Logics over Traces

Bharat Adsul* and Milind Sohoni

Department of Computer Science and Engineering
Indian Institute of Technology, Mumbai 400 076, India
{abharat,sohoni}@cse.iitb.ac.in
Fax: 91 22 572 0290

Abstract. We investigate local and global paradigms of reasoning about distributed behaviours, modelled as Mazurkiewicz traces, in the context of first-order and monadic second-order logics. We describe new normal forms for properties expressible in these logics. The first normal form, surprisingly, yields a decomposition of a global property as a boolean combination of local properties. The second normal form strengthens McNaughton's theorem and states that global properties of infinite behaviours may also be described as boolean combinations of recurring properties of finite local histories of the behaviours. We briefly touch upon some of the interesting applications of these normal forms.

Introduction

This paper is about the behaviours of distributed systems. These systems may be described as a finite set of processes (agents) which hand-shake to exchange information and thus coordinate their behaviour. The correct models for their behaviours are Mazurkiewicz traces or simply traces (see [DR]). This is a very special class of labelled partial orders wherein agents, their actions (events) and causality between events are clearly understood. Furthermore, every linearization of this partial order is a possible global trajectory of the distributed system. As such, distributed systems may be regarded "globally" as sequential systems and first-order (FO) and monadic second-order (MSO) logics over sequences may be used to analyse them. However, these logics, though rich in expressivity, ignore the causality relationship between events in a distributed system: events which occur consequentially, may not be causally related at all.

It is possible to naturally extend the logical framework of FO and MSO to traces which "equals" the sequential FO and MSO respectively in expressivity [EM]. These logics may explicitly talk of causality relationship among events of a distributed behaviour. However, an important intricacy of a distributed system is the notion of a *local view* of a behaviour for an agent, which corresponds to those events in the behaviour of the system which this agent has "seen". These local views provide a clear understanding to each agent about the unfolding of the global behaviour. It is natural to have formulas located at agents which may only assert properties of that agent's view of the global behaviour. Such semantics are natural for they correspond exactly to those properties of the behaviour

* The work of this author was supported by an Infosys Fellowship.

which are "observable" internally to the agents. We propose DFO and DMSO, viz., distributed FO and MSO, natural variants of FO and MSO over traces, which allow formulation of local properties and their boolean combinations. The natural question then is, whether DFO and DMSO equal FO and MSO in expressive power respectively. Surprisingly, the answer to this question is indeed affirmative. This is shown by obtaining what we call the *first normal form* for a FO/MSO formula into an equivalent DFO/DMSO formula.

The second half of the paper presents a strengthening of the fundamental theorem of McNaughton (see [Tho2]) to traces. We introduce local limits of languages of finite traces. As expected, membership within a local limit is decided by the local views of a fixed agent. We then arrive at the *second normal form* which asserts that FO-definable (MSO-definable) languages of infinite traces are boolean combinations of local limits of FO-definable (respectively MSO-definable) languages of finite traces.

An outline of the paper is as follows. In the next section we develop notations for traces and setup the first-order framework and its variants. In Section 2, we prove the first normal form for the first-order system. We present the second normal form for the first-order system in Section 3. The proof is based on the first normal form and an intermediate normal form. This intermediate normal form is motivated by a similar normal form for the monadic second-order system [DM]. Section 4 deals with the normal forms for the monadic second-order system. After presenting these results we conclude in Section 5 by giving new applications of the normal forms presented in this paper. We close this section with a remark that the normal forms have already provided a key argument in showing expressive completeness for a new temporal logic over traces [AS1].

1 Basic Notations

Traces. Let $\mathcal{P} = \{1, 2, \ldots, K\}$ be a set of processes. A distributed alphabet over $\mathcal{P}$ is a family $\widetilde{\Sigma} = (\Sigma_i)_{i\in\mathcal{P}}$. Let $\Sigma = \bigcup_{i\in\mathcal{P}} \Sigma_i$. For $a \in \Sigma$, we set $\mathrm{loc}(a) = \{i \in \mathcal{P} \mid a \in \Sigma_i\}$. By (Σ, I) we denote the corresponding trace alphabet, i.e., I is the independence relation $I = \{(a, b) \in \Sigma^2 \mid \mathrm{loc}(a) \cap \mathrm{loc}(b) = \emptyset\}$ induced by $\widetilde{\Sigma}$. The corresponding dependence relation $\Sigma^2 \setminus I$ is denoted by D.

A Σ-labelled poset is a structure $F = (E, \leq, \lambda)$, where E is a set, $\leq$ is a partial order on E and $\lambda : E \to \Sigma$ is a labelling function. For $e, e' \in E$, define $e \lessdot e'$ iff $e < e'$ and for each e'' with $e \leq e'' \leq e'$ either $e = e''$ or $e' = e''$. For $X \subseteq E$, let $\downarrow X = \{y \in E \mid y \leq x$ for some $x \in X\}$. For $e \in E$, we set $\downarrow e = \downarrow\{e\}$. A trace over $\widetilde{\Sigma}$ is a countable Σ-labelled poset $F = (E, \leq, \lambda)$ satisfying the following conditions.

- For each $e \in E$, $\downarrow e$ is a finite set.
- If $e, e' \in E$ with $e \lessdot e'$ then $(\lambda(e), \lambda(e')) \in D$.
- If $e, e' \in E$ with $(\lambda(e), \lambda(e')) \in D$ then $e \leq e'$ or $e' \leq e$.

Let $TR(\widetilde{\Sigma})$ denote the set of traces over $\widetilde{\Sigma}$. Fix $F = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. We say F is a finite trace if E is finite, otherwise F is said to be an infinite trace.

We denote sets of finite and infinite traces over $\widetilde{\Sigma}$ by $TR^*(\widetilde{\Sigma})$ and $TR^\omega(\widetilde{\Sigma})$ respectively. Henceforth, a trace means a trace over $\widetilde{\Sigma}$ unless specified otherwise.

Let $F = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. E is the set of events in F and for an event e in F, $\mathrm{loc}(e)$ abbreviates $\mathrm{loc}(\lambda(e))$. Further, let $i \in \mathcal{P}$. The set of i-events in F is $E_i = \{e \in E \mid i \in \mathrm{loc}(e)\}$. This is the set of events in which process i participates. It is clear that E_i is totally ordered with respect to $\leq$.

We let F_i denote the induced trace $(\downarrow E_i, \leq_i, \lambda_i)$ where $\leq_i = \leq \cap (\downarrow E_i \times \downarrow E_i)$ and λ_i is the restriction of λ to $\downarrow E_i$. The trace F_i represents the knowledge of process i about the trace F. For $Q \subseteq \mathcal{P}$, let F_Q denote the induced trace $(\cup_{i \in Q} \downarrow E_i, \leq_Q, \lambda_Q)$ where $\leq_Q$ and λ_Q are restrictions of $\leq$ and λ to $\cup_{i \in Q} \downarrow E_i$ respectively. The trace F_Q represents the collective knowledge of the processes in Q about the trace F. An easy observation is that F_Q is completely determined by the collection $\{F_i\}_{i \in Q}$. Also, note that $F_{\mathcal{P}} = F$.

First-Order Theories. We fix a countable set of individual variables X and let x, y with or without subscripts range over X.

We now turn to $\mathrm{FO}(\widetilde{\Sigma})$, the global first-order theory of traces over $\widetilde{\Sigma}$. For each $a \in \Sigma$, there is a unary predicate symbol R_a. There is also a binary predicate symbol $\leq$. $R_a(x)$ and $x \leq y$ are atomic formulas. If φ and φ' are $\mathrm{FO}(\widetilde{\Sigma})$ formulas, so are $\neg\varphi$, $\varphi \vee \varphi'$ and $(\exists x)\varphi$.

As structures for this global first-order theory, we limit ourselves to elements of $TR(\widetilde{\Sigma})$. Let $F \in TR(\widetilde{\Sigma})$ with $F = (E, \leq, \lambda)$ and let $\mathcal{I} : X \to E$ be an interpretation. Then $F \models_{\mathcal{I}} R_a(x)$ iff $\lambda(\mathcal{I}(x)) = a$ and $F \models_{\mathcal{I}} x \leq y$ iff $\mathcal{I}(x) \leq \mathcal{I}(y)$. The remaining semantic definitions go along the expected lines. In other words, we work with the finite relational signature $\mathcal{S} = (\leq, \{R_a\}_{a \in \Sigma})$ where $\leq$ is a binary relation and $\{R_a\}_{a \in \Sigma}$ is a family of unary relations. For a sentence φ of $\mathrm{FO}(\widetilde{\Sigma})$, we write $F \models \varphi$ with the intended meaning that F is a model of φ.

Now we introduce local first-order theories. We associate a local first-order theory, $\mathrm{FO}_i(\widetilde{\Sigma})$, for each $i \in \mathcal{P}$. Let $i \in \mathcal{P}$. It is easy to describe the sentences of $\mathrm{FO}_i(\widetilde{\Sigma})$. Let φ be a sentence of $\mathrm{FO}(\widetilde{\Sigma})$. Then $\varphi[i]$ is a sentence of $\mathrm{FO}_i(\widetilde{\Sigma})$.

The structures of $\mathrm{FO}_i(\widetilde{\Sigma})$ are exactly the same as that of $\mathrm{FO}(\widetilde{\Sigma})$. Let $F \in TR(\widetilde{\Sigma})$ with $F = (E, \leq, \lambda)$. The models relation $\models_i$ of $\mathrm{FO}_i(\widetilde{\Sigma})$ is defined as: $F \models_i \varphi[i]$ if $F_i \models \varphi$. This clarifies our usage *local* first-order theory for $\mathrm{FO}_i(\widetilde{\Sigma})$. For notational convenience we denote $\models_i$ also by $\models$.

Now we define the distributed first-order theory $\mathrm{DFO}(\widetilde{\Sigma})$. If φ is a formula of $\mathrm{FO}_i(\widetilde{\Sigma})$ for some $i \in \mathcal{P}$ then φ is also a formula of $\mathrm{DFO}(\widetilde{\Sigma})$. In this case, we say that φ is an i-type formula of $\mathrm{DFO}(\widetilde{\Sigma})$. If φ and φ' are formulas of $\mathrm{DFO}(\widetilde{\Sigma})$, so are $\varphi \wedge \varphi'$ and $\varphi \vee \varphi'$. The models relation for $\mathrm{DFO}(\widetilde{\Sigma})$ is defined in an obvious manner making use of the models relations of local first-order theories. By further abuse of notation, we denote this models relation also by $\models$.

2 First Normal Form

We now describe the first normal form for $\mathrm{FO}(\widetilde{\Sigma})$. We say sentences φ and φ' (of possibly different theories) are equivalent, denoted $\varphi = \varphi'$, if $F \models \varphi$ iff $F \models \varphi'$

for every $F \in TR(\widetilde{\Sigma})$. Two theories are said to be equivalent if every sentence of one theory is equivalent to some sentence of the other theory and vice versa.

Theorem 1 (First Normal Form). *The global first-order theory* $\mathrm{FO}(\widetilde{\Sigma})$ *is equivalent to the distributed first-order theory* $\mathrm{DFO}(\widetilde{\Sigma})$.

Given a sentence φ of $\mathrm{DFO}(\widetilde{\Sigma})$, it is easy to construct an equivalent $\mathrm{FO}(\widetilde{\Sigma})$ sentence. The basic trick is to translate, inductively, every quantification $\exists x$ of a $\mathrm{FO}_i(\widetilde{\Sigma})$ variable x in φ into $\exists x \exists y (\vee_{a \in \Sigma_i} R_a(y)) \wedge (x \leq y)$. This indicates the proof of one direction of Theorem 1.

We formulate the other direction of the above theorem using the Ehrenfeucht-Fraïssé game technique (see [EF]). We start with some notations. We denote by $\mathrm{qd}(\varphi)$ the quantifier depth of a sentence φ. It is defined to be the maximum number of nested quantifiers in φ. Let n be a non-negative integer. For $F \in TR(\widetilde{\Sigma})$, we denote the n-theory of F by $\mathrm{Th}_n(F)$ where

$$\mathrm{Th}_n(F) = \{\varphi \mid \varphi \text{ is a sentence of } \mathrm{FO}(\widetilde{\Sigma}) \text{ with } \mathrm{qd}(\varphi) \leq n \text{ and } F \models \varphi\}$$

Proposition 1. *There exists a positive integer κ depending only on $\widetilde{\Sigma}$ such that, for $n \geq 0$ and $F \in TR(\widetilde{\Sigma})$, the n-theory* $\mathrm{Th}_n(F)$ *is determined by the collection* $\{\mathrm{Th}_{n+\kappa}(F_i)\}_{i \in \mathcal{P}}$ *of $n{+}\kappa$-theories.*

Now we argue that Proposition 1 implies Theorem 1. We first recall some standard results of logic [EF]. Let $\mathrm{TH}_n = \{\mathrm{Th}_n(F) \mid F \in TR(\widetilde{\Sigma})\}$ be the set of all n-theories. An easy induction on n shows that TH_n is a finite collection. For each n-theory there is a Hintikka sentence of quantifier depth n which characterizes it. More precisely, suppose T is an n-theory and φ_T is the corresponding Hintikka sentence. Then, for $F \in TR(\widetilde{\Sigma})$, $\mathrm{Th}_n(F) = T$ iff $F \models \varphi_T$. Moreover, every sentence of quantifier depth n can be written as a disjunction of those Hintikka sentences of quantifier depth n from which it follows. Note that there are only finitely many Hintikka sentences of quantifier depth n.

Recall that, in order to prove the remaining part of Theorem 1, we have to show the following: given a sentence φ of $\mathrm{FO}(\widetilde{\Sigma})$, there exists an equivalent sentence of $\mathrm{DFO}(\widetilde{\Sigma})$. In view of the above discussion, we may assume that φ is a Hintikka sentence φ_T of quantifier depth n for some $n \geq 0$. We say a collection $\{T_i\}_{i \in \mathcal{P}}$ of $n{+}\kappa$-theories *implies* the n-theory T iff there exists $F \in TR(\widetilde{\Sigma})$ such that $\mathrm{Th}_n(F) = T$ and $\mathrm{Th}_{n+\kappa}(F_i) = T_i$ for each $i \in \mathcal{P}$. Apriori $\{T_i\}_{i \in \mathcal{P}}$ may imply T as well as T' with $T \neq T'$. Proposition 1 asserts that this is not possible. We set $\mathcal{T} = \{\{T_i\}_{i \in \mathcal{P}} \mid \{T_i\}_{i \in \mathcal{P}} \text{ implies } T\}$. Observe that $\mathcal{T}$ is finite. Assuming Proposition 1, it is not very difficult to see that $\varphi_T = \vee_{\{T_i\} \in \mathcal{T}} \ \wedge_{i \in \mathcal{P}} \varphi_{T_i}[i]$ where φ_{T_i} is the Hintikka sentence of $\mathrm{FO}(\widetilde{\Sigma})$ corresponding to T_i. This gives an equivalent sentence of $\mathrm{DFO}(\widetilde{\Sigma})$ for φ_T completing the proof of the remaining part of Theorem 1.

Note that, as the above argument shows, even the following weak form of Proposition 1 suffices to prove the theorem: for each n, there exists $\kappa(n)$ such that, for $F \in TR(\widetilde{\Sigma})$, the collection $\{\mathrm{Th}_{\kappa(n)}(F_i)\}_{i \in \mathcal{P}}$ determines $\mathrm{Th}_n(F)$. It is natural to ask *if there exist $\kappa(n)$, for each n, such that, for $F \in TR(\widetilde{\Sigma})$, the collection* $\{\mathrm{Th}_{\kappa(n)}(F_i)\}_{i \in Q}$ *determines* $\mathrm{Th}_n(F_Q)$ for an arbitrary subset $Q \subseteq \mathcal{P}$. The next example answers this question negatively already for $n = 2$. It also highlights the importance of distribution. We will see later in Section 4 that the above question is answered in affirmative for the monadic second-order theory.

Example 1. Let $\widetilde{\Pi} = (\Pi_1, \Pi_2, \Pi_3)$ be a distributed alphabet over three processes where $\Pi_1 = \{a\}, \Pi_2 = \{b\}, \Pi_3 = \{a, b\}$. Observe that, with $\Pi = \bigcup \Pi_i = \{a, b\}$, $TR(\widetilde{\Pi}) = \Pi^*$. Keeping this in mind, we write a trace over $\widetilde{\Pi}$ as a word over Π. Let $Q = \{1, 2\}$. Note that, for $F \in TR(\widetilde{\Pi})$, $F_Q = F$.

Now we show that, for every m, there exist two traces F and H in $TR(\widetilde{\Pi})$ such that $\mathrm{Th}_m(F_1) = \mathrm{Th}_m(H_1), \mathrm{Th}_m(F_2) = \mathrm{Th}_m(H_2)$ but $\mathrm{Th}_2(F_Q) \neq \mathrm{Th}_2(H_Q)$.

Let $F = (ab)^{2^m - 1}a$ and $H = (ab)^{2^m}$. It is easy to check that $F_1 = H_1 = F$, $F_2 = (ab)^{2^m - 1}$ and $H_2 = (ab)^{2^m}$. Clearly, $\mathrm{Th}_m(F_1) = \mathrm{Th}_m(H_1)$. It is not very difficult to show that $\mathrm{Th}_m(F_2) = \mathrm{Th}_m(H_2)$ (see [Tho2]).

As F and H have different letters in the last position, $\mathrm{Th}_2(F) \neq \mathrm{Th}_2(H)$. Since, $F_Q = F$ and $H_Q = H$, $\mathrm{Th}_2(F_Q) \neq \mathrm{Th}_2(H_Q)$.

Ehrenfeucht-Fraïssé Game Formulation. The proof of Proposition 1 involves an application of the Ehrenfeucht-Fraïssé game, EF-game. Fix $F, H \in TR(\widetilde{\Sigma})$ with $F = (E, \leq, \lambda)$ and $H = (G, \leq, \mu)$. We refer to E and G as universes for the structures F and H respectively. We denote the n-round EF-game by $\mathcal{G}_n(F, H)$. The n-round game $\mathcal{G}_n(F, H)$ is played between two players called Spoiler and Duplicator. In round m, Spoiler picks an element e_m in E or an element g_m in G. Duplicator responds by picking an element in the other universe, that is, g_m in G, respectively e_m in E. After the n rounds, Duplicator wins the play if atomic formulas cannot distinguish between $(e_m)_{1 \leq m \leq n}$ and $(g_m)_{1 \leq m \leq n}$. More precisely, we should have

1. For $1 \leq m \leq n$ and $a \in \Sigma$, $R_a(e_m)$ iff $R_a(g_m)$ (i.e., $\lambda(e_m) = a$ iff $\mu(g_m) = a$).
2. For $1 \leq m, m' \leq n$, $e_m \leq e_{m'}$ iff $g_m \leq g_{m'}$.

We write $F \sim_n H$ if Duplicator has a winning strategy in $\mathcal{G}_n(F, H)$.

Now we invoke the Ehrenfeucht-Fraïssé theorem which says that, for $F, H \in TR(\widetilde{\Sigma})$ and $n \geq 0$, $\mathrm{Th}_n(F) = \mathrm{Th}_n(H)$ iff $F \sim_n H$. Therefore, in order to prove the above proposition, it suffices to prove the following claim.

Claim 1. *There exists an integer κ such that the following holds. Let $n \geq 0$ and $F, H \in TR(\widetilde{\Sigma})$ with $F = (E, \leq, \lambda)$ and $H = (G, \leq, \mu)$. Suppose that for each $i \in \mathcal{P}$, $F_i \sim_{n+\kappa} H_i$. Then $F \sim_n H$. In other words, if Duplicator has winning strategies in the local games $\{\mathcal{G}_{n+\kappa}(F_i, H_i)\}_{i \in \mathcal{P}}$, then it also has a winning strategy in the global game $\mathcal{G}_n(F, H)$.*

For the sake of simplicity and clarity, we will prove the claim only for finite traces. The proof goes through even for infinite traces.

We prove the claim by actually constructing a winning strategy in the global game by patching together the winning strategies in the local games. Spoiler's move in a typical round of a play in the global game is copied to one of the local games. Duplicator's response there is then copied back to the global game. In general, Duplicator's responses in different rounds are copied from different local games. In order to relate these "independent" responses from different local games, we need some technical machinery and we develop that now.

Latest Information. Fix $F = (E, \leq, \lambda) \in TR^*(\widetilde{\Sigma})$. Let $i, j, k \in \mathcal{P}$. Following [MS], we define $\mathsf{latest}_{i \to j}(F) = e$ where e is the maximum j-event in $\downarrow E_i$. If e does not exist then $\mathsf{latest}_{i \to j}(F)$ is undefined. Further, we define $\mathsf{latest}_{i \to j \to k}(F)$ to be the maximum k-event in $\downarrow \mathsf{latest}_{i \to j}(F)$. We set $\mathsf{primary}_i(F) = \cup_{l \in \mathcal{P}} \{\mathsf{latest}_{i \to l}(F)\}$ and $\mathsf{secondary}_i(F) = \cup_{l,m \in \mathcal{P}} \{\mathsf{latest}_{i \to l \to m}(F)\}$. As $\mathsf{latest}_{i \to j}(F) = \mathsf{latest}_{i \to j \to j}(F)$, $\mathsf{primary}_i(F) \subseteq \mathsf{secondary}_i(F)$. Observe that $|\mathsf{secondary}_i(F)| \leq K^2$.

The next two lemmas (see [AS2]) are the key to the proof of Claim 1.

Lemma 1. *Let $e, e' \in E$ with $e \in \downarrow E_i$ and $e' \in (\downarrow E_j \setminus \downarrow E_i)$ for some $i, j \in \mathcal{P}$. Then there exists $e'' \in \mathsf{primary}_i(F) \cap \mathsf{secondary}_j(F)$ such that $e \leq e'$ iff $e \leq e'' \leq e'$. Moreover, there exists $k, l \in \mathcal{P}$ such that $e'' = \mathsf{latest}_{i \to k}(F) = \mathsf{latest}_{j \to l \to k}(F)$.*

Lemma 2. *There exists an integer κ with the following property. Let $F, H \in TR^*(\widetilde{\Sigma})$. Fix $m \geq \kappa$. Suppose that, for each $i \in \mathcal{P}$, $F_i \sim_m H_i$. Then, for all $i, j, k, l \in \mathcal{P}$, $\mathsf{latest}_{i \to k}(F) = \mathsf{latest}_{j \to l \to k}(F)$ iff $\mathsf{latest}_{i \to k}(H) = \mathsf{latest}_{j \to l \to k}(H)$.*

Construction of Global Strategy Using Local Strategies — Proof of Claim 1. Let $n \geq 0$. Fix κ as stated in Lemma 2. We may assume that $\kappa \geq K^2$.

For each $i \in \mathcal{P}$, we fix a winning strategy for Duplicator in the $n+\kappa$-round game $\mathcal{G}_{n+\kappa}(F_i, H_i)$. By our assumption, $F_i \sim_{n+\kappa} H_i$, such a strategy exists.

Recall that our aim is to construct a winning strategy for Duplicator in the global game $\mathcal{G}_n(F, H)$. Towards this, we first fix a number (atmost K^2) of initial rounds in the local games and then start constructing the global strategy by copying Spoiler's moves in the global game to appropriate local games where these fixed initial rounds are already played.

Fix $i \in \mathcal{P}$. Now we describe the initial $|\mathsf{secondary}_i(F)|$ rounds of the game $\mathcal{G}_{n+\kappa}(F_i, H_i)$ where the moves of Spoiler are fixed.

We arbitrarily order the elements in $\mathsf{secondary}_i(F)$. In round m, Spoiler picks the m-th element in this order in $\downarrow E_i$ which is the universe of F_i. Duplicator responds using the fixed winning strategy. Suppose in the m-th round, Spoiler picks $\mathsf{latest}_{i \to j \to k}(F)$. Then it is easy to see that Duplicator must also pick the corresponding element in $\mathsf{secondary}_i(H)$, that is, $\mathsf{latest}_{i \to j \to k}(H)$, in $\downarrow G_i$ which is the universe of H_i in order to ensure a win with the fixed winning strategy.

Henceforth we assume that, for each $i \in \mathcal{P}$, the above described $|\mathsf{secondary}_i(F)|$ initial rounds of the game $\mathcal{G}_{n+\kappa}(F_i, H_i)$ have already been played.

Now consider the n-round game $\mathcal{G}_n(F, H)$. Suppose that in the m-th round Spoiler picks e_m in E in the game $\mathcal{G}_n(F, H)$. Let i be the least index in $\mathcal{P}$ such that $e_m \in \downarrow E_i$. We copy Spoiler's move to the (partially played) game $\mathcal{G}_{n+\kappa}(F_i, H_i)$. That is, we play another additional round in the game $\mathcal{G}_{n+\kappa}(F_i, H_i)$ with Spoiler picking e_m in $\downarrow E_i$, the universe of F_i. Duplicator responds to this move in $\mathcal{G}_{n+\kappa}(F_i, H_i)$ with an element g_m in $\downarrow G_i$ using the winning strategy fixed earlier. Now we copy this move to the game $\mathcal{G}_n(F, H)$. That is, in the m-th round of $\mathcal{G}_n(F, H)$, Duplicator responds with g_m.

Instead if Spoiler picks g_m in G in the m-th round of $\mathcal{G}_n(F, H)$, then Duplicator's response e_m in E is determined similarly.

Observe that a play of n rounds on the pair (F, H) induces plays of atmost $n+K^2$ rounds on the pairs (F_i, G_i).

Now we prove that the strategy for Duplicator described above in the game $\mathcal{G}_n(F, H)$ is in fact a winning strategy. We verify that the two conditions ensuring a win for Duplicator indeed hold. The first condition is easy to verify and we skip the details. Now let e_m, g_m and $e_{m'}, g_{m'}$ be the elements picked in the m-th and m'-th round of the game $\mathcal{G}_n(F, H)$, respectively. Further, let $i, j \in \mathcal{P}$ be such that in the m-th and m'-th rounds, Duplicator copied the moves as suggested in the induced plays of the games $\mathcal{G}_{n+\kappa}(F_i, H_i)$ and $\mathcal{G}_{n+\kappa}(F_j, H_j)$ respectively. If $i = j$ then $e_m \leq e_{m'}$ iff $g_m \leq g_{m'}$ because then Duplicator copied the winning strategy in the game $\mathcal{G}_{n+\kappa}(F_i, H_i)$.

Now we deal with the most interesting case where i is not equal to j. Suppose $e_m \leq e_{m'}$. It is easy to see that $e_{m'} \notin \downarrow E_i$. By Lemma 1, there exists $e \in \mathsf{primary}_i(F) \cap \mathsf{secondary}_j(F)$ such that $e_m \leq e$ and $e \leq e_{m'}$. Let $k, l \in \mathcal{P}$ be such that $\mathsf{latest}_{i \to k}(F) = \mathsf{latest}_{j \to l \to k}(F) = e$. By Lemma 2, $\mathsf{latest}_{i \to k}(H) = \mathsf{latest}_{j \to l \to k}(H)$ (say g), as $F_i \sim_{n+\kappa} H_i$ for each $i \in \mathcal{P}$.

Note that e, being a member of $\mathsf{secondary}_i(F)$ and $\mathsf{secondary}_j(F)$, was picked in the initial $|\mathsf{secondary}_i(F)|$ rounds of the game $\mathcal{G}_{n+\kappa}(F_i, H_i)$ as well as in the initial $|\mathsf{secondary}_j(F)|$ rounds of the game $\mathcal{G}_{n+\kappa}(F_j, H_j)$. As noted earlier, g must be the response to the respective moves. Thus, pairs $(e, g), (e_m, g_m)$ come from the induced play in the game $\mathcal{G}_{n+\kappa}(F_i, H_i)$ and pairs $(e, g), (e_{m'}, g_{m'})$ come from the induced play in the game $\mathcal{G}_{n+\kappa}(F_j, H_j)$. As Duplicator's moves were determined by the winning strategies in the games $\mathcal{G}_{n+\kappa}(F_i, H_i)$ and $\mathcal{G}_{n+\kappa}(F_j, H_j)$, it follows that $g_m \leq g$ and $g \leq g_{m'}$. Therefore $g_m \leq g_{m'}$. The other direction can be shown similarly. $\square$

3 Second Normal Form

This section is concerned with the local trace analogue of McNaughton's theorem. We formulate this as an equivalence of two theories – FO($\widetilde{\Sigma}$) and DLFO($\widetilde{\Sigma}$). DLFO($\widetilde{\Sigma}$) is equipped with local limits to assert FO($\widetilde{\Sigma}$) recurring properties of local views of processes. We begin by describing the formulas of DLFO($\widetilde{\Sigma}$). Let φ be a formula of FO($\widetilde{\Sigma}$). Then for each $i \in \mathcal{P}$, $\lim_i^*(\varphi)$ and $\lim_i^\omega(\varphi)$ are formulas of DLFO($\widetilde{\Sigma}$). Further, if φ and φ' are formulas of DLFO($\widetilde{\Sigma}$), so are $\neg\varphi, \varphi \vee \varphi'$. We refer to DLFO($\widetilde{\Sigma}$) as the distributed limit first-order theory.

Let $F = (E, \leq, \lambda) \in TR(\widetilde{\Sigma})$. Then $c \subseteq E$ is a configuration iff c is finite and $\downarrow c = c$. We let $\mathcal{C}_F$ denote the set of configurations of F. Notice that the empty set is a configuration. More importantly, $\downarrow e$ is a configuration for every $e \in E$.

Recall that $E_i = \{e \mid e \in E \text{ and } i \in \mathrm{loc}(e)\}$. Let $c \in \mathcal{C}_F$ and $i \in \mathcal{P}$. Then $\downarrow^i(c)$ is the i-view of c and it is defined as: $\downarrow^i(c) = \downarrow(c \cap E_i)$. We note that $\downarrow^i(c)$ is also a configuration. It is the "best" configuration that the process i is aware of at c. It is easy to see that if $\downarrow^i(c) \neq \emptyset$ then there exists $e \in E_i$ such that $\downarrow^i(c) = \downarrow e$. We say that $c \in \mathcal{C}_F$ is an i-local configuration if $c = \downarrow^i(c)$. Let $\mathcal{C}_F^i$ denote the set of i-local configurations. For $Q \subseteq \mathcal{P}$ and $c \in \mathcal{C}_F$, we let $\downarrow^Q(c)$ denote the set $\bigcup\{\downarrow^i(c) \mid i \in Q\}$. Once again, $\downarrow^Q(c)$ is a configuration. It represents the collective knowledge of the processes in Q about the configuration c.

Let $c \in \mathcal{C}_F$. We denote by F_c the induced finite trace $F_c = (c, \leq', \lambda') \in TR^*(\widetilde{\Sigma})$ where $\leq' = \leq \cap (c \times c)$ and $\lambda' : c \to \Sigma$ is λ restricted to c.

As structures for DLFO$(\widetilde{\Sigma})$, we confine ourselves to infinite traces. Let $F \in TR^\omega(\widetilde{\Sigma})$ with $F = (E, \leq, \lambda)$. Now we are ready to define the models relation of DLFO$(\widetilde{\Sigma})$ using the models relation of FO$(\widetilde{\Sigma})$ in the following manner.

- $F \models \neg\varphi$ iff $F \not\models \varphi$.
- $F \models \varphi \vee \varphi'$ iff $F \models \varphi$ or $F \models \varphi'$.
- $F \models \lim_i^*(\varphi)$ iff E_i is finite and with $c = \downarrow E_i$, $F_c \models \varphi$.
- $F \models \lim_i^\omega(\varphi)$ iff E_i is infinite and there are infinitely many i-local configurations c such that $F_c \models \varphi$.

The assertion $\lim_i^*(\varphi)$ says that process i participates in a finite number of events. Moreover, at the end, φ holds in its view. On the other hand, $\lim_i^\omega(\varphi)$ says that process i participates in an infinite number of events and infinitely many local views of i assert φ. An easy consequence of this is that $F \models \lim_i^*(\varphi)$ iff $F_i \models \lim_i^*(\varphi)$. Similarly, $F \models \lim_i^\omega(\varphi)$ iff $F_i \models \lim_i^\omega(\varphi)$.

Theorem 2 (Second Normal Form). *The global first-order theory* FO$(\widetilde{\Sigma})$ *over infinite traces is equivalent to the distributed limit first-order theory* DLFO$(\widetilde{\Sigma})$.

It is easy to see that for each sentence φ of DLFO$(\widetilde{\Sigma})$ there is an equivalent FO$(\widetilde{\Sigma})$ sentence. This amounts to showing that for a sentence φ of FO$(\widetilde{\Sigma})$ there exist sentences φ' and φ'' of FO$(\widetilde{\Sigma})$ which are equivalent to the sentences $\lim_i^*(\varphi)$ and $\lim_i^\omega(\varphi)$ of DLFO$(\widetilde{\Sigma})$. We omit the details of this straightforward translation.

Before we proceed to prove the other direction of Theorem 2, we need to introduce some trace language-theoretic notions. A trace language is just a subset of $TR(\widetilde{\Sigma})$. A sentence φ of FO$(\widetilde{\Sigma})$ naturally induces the trace languages $L_\varphi^\omega = \{F \in TR^\omega(\widetilde{\Sigma}) \mid F \models \varphi\}$ and $L_\varphi^* = \{F \in TR^*(\widetilde{\Sigma}) \mid F \models \varphi\}$. Similarly, a sentence φ of DLFO$(\widetilde{\Sigma})$ naturally induces the language $L_\varphi = \{F \in TR^\omega(\widetilde{\Sigma}) \mid F \models \varphi\}$. We say a language $L \subseteq TR^*(\widetilde{\Sigma})$ is FO$(\widetilde{\Sigma})$-definable if there exists a sentence φ of FO$(\widetilde{\Sigma})$ such that $L = L_\varphi^*$. We define the notion of FO$(\widetilde{\Sigma})$-definability and DLFO$(\widetilde{\Sigma})$-definability for languages of $TR^\omega(\widetilde{\Sigma})$ along the same lines. In this new notation, we may phrase the remaining part of Theorem 2 as follows.

Claim 2. *Let $L \subseteq TR^\omega(\widetilde{\Sigma})$ be* FO$(\widetilde{\Sigma})$*-definable. Then L is* DLFO$(\widetilde{\Sigma})$*-definable.*

The claim is proved in two steps. In the first step (Proposition 2) we give an intermediate normal form for FO$(\widetilde{\Sigma})$-definable trace languages of $TR^\omega(\widetilde{\Sigma})$.

Finally, the first normal form is crucially applied to the intermediate normal form to arrive at the required result (Lemma 4).

We start with the following notations. For $F = (E, \leq, \lambda) \in TR^*(\widetilde{\Sigma})$, we set $\mathrm{alph}(F) = \lambda(E)$. For $F \in TR^\omega(\widetilde{\Sigma})$ with $F = (E, \leq, \lambda)$, we set $\mathrm{alphinf}(F) = \{a \in \Sigma \mid \text{ for infinitely many } e \in E, \lambda(e) = a\}$. Let $A \subseteq \Sigma$ and $TR_A \subseteq TR^\omega(\widetilde{\Sigma})$ be defined as: $TR_A = \{F \in TR^\omega(\widetilde{\Sigma}) \mid \mathrm{alphinf}(F) = A\}$.

With A, we associate an undirected graph $\mathcal{G}_A$ whose vertex set is A and there is an edge between a and b in A if $(a, b) \notin I$. Let $\mathcal{C}_A$ denote the maximal connected components of $\mathcal{G}_A$. We think of $\mathcal{C}_A$ as a partition of A. Let $\mathcal{C}_A = \{A_1, A_2, \ldots, A_k\}$. For each $1 \leq m \leq k$, set $P_m = \{j \in \mathcal{P} \mid \Sigma_j \cap A_m \neq \emptyset\}$ and j_m to be the lexicographically least member of P_m. Further, set $Q = \{j \in \mathcal{P} \mid \Sigma_j \cap A = \emptyset\}$. Note that $\mathcal{P} = P_1 \cup P_2 \ldots P_k \cup Q$. We let $\mathcal{P}_A = Q \cup \{j_1, j_2, \ldots, j_k\} \subseteq \mathcal{P}$.

Let $F = (E, \leq, \lambda) \in TR^*(\widetilde{\Sigma})$. We say F is $\mathcal{P}_A$-pointed if $\downarrow^{P_m}(E) = \downarrow^{j_m}(E)$ for each $1 \leq m \leq k$. In other words, for $1 \leq m \leq k$ and $j \in P_m$, $\downarrow^j(E) = \downarrow^j(\downarrow^{j_m}(E))$.

A basic observation regarding traces in TR_A is captured in the next lemma.

Lemma 3. *Let $F = (E, \leq, \lambda) \in TR_A$. There is, for each $1 \leq m \leq k$, a j_m-event e_m such that the following holds. For each $1 \leq m \leq k$, let f_m be a j_m-event such that $e_m \leq f_m$. Further, let $c \in \mathcal{C}_F$ be defined as $c = \cup_{j \in Q}(\downarrow E_j) \cup_{1 \leq m \leq k} (\downarrow f_m)$. Then the induced finite trace F_c is $\mathcal{P}_A$-pointed.*

Let $J \subseteq TR^*(\widetilde{\Sigma})$. We say J is $\mathcal{P}_A$-pointed if every trace in J is so. We next define $\lim(J) \subseteq TR^\omega(\widetilde{\Sigma})$ which is crucially used in the intermediate normal form.

$$\lim(J) = \left\{ F = (E, \leq, \lambda) \,\middle|\, \begin{array}{l} \text{there exists an infinite chain } c_1 \subseteq c_2 \ldots \text{ in } \mathcal{C}_F \\ \text{such that, for all } i > 0, F_{c_i} \in J \text{ and } E = \bigcup_i c_i \end{array} \right\}$$

Proposition 2 (Intermediate Normal Form). *The class of* $\mathrm{FO}(\widetilde{\Sigma})$*-definable trace languages of $TR^\omega(\widetilde{\Sigma})$ equals the boolean closure of the family of languages of type* $\lim(J) \cap TR_A$*, where $A \subseteq \Sigma$ and J is a $\mathcal{P}_A$-pointed* $\mathrm{FO}(\widetilde{\Sigma})$*-definable trace language of $TR^*(\widetilde{\Sigma})$.*

A similar characterization for $\mathrm{MSO}(\widetilde{\Sigma})$-definable ($\omega$-regular) trace languages of $TR^\omega(\widetilde{\Sigma})$ was already proved by Diekert and Muscholl in [DM]. Our proof (see [AS2]) relies on a careful analysis of the steps involved in [DM] and other characterizations of $\mathrm{FO}(\widetilde{\Sigma})$-definable trace languages [EM].

Now we get back to the proof of Claim 2. Let $A \subseteq \Sigma$ and J be a $\mathcal{P}_A$-pointed $\mathrm{FO}(\widetilde{\Sigma})$-definable trace language of $TR^*(\widetilde{\Sigma})$. It suffices to show that $\lim(J) \cap TR_A$ is $\mathrm{DLFO}(\widetilde{\Sigma})$-definable. It is easy to see that TR_A is $\mathrm{DLFO}(\widetilde{\Sigma})$-definable. Let φ_A be a sentence of $\mathrm{DLFO}(\widetilde{\Sigma})$ such that $TR_A = L_{\varphi_A}$.

Let φ be a sentence of $\mathrm{FO}(\widetilde{\Sigma})$ such that $J = L^*_\varphi$. By the first normal form, there exists an $n \times K$-array $\{\varphi_{ij}\}_{1 \leq i \leq n, j \in \mathcal{P}}$ of sentences of $\mathrm{FO}(\widetilde{\Sigma})$ such that $\varphi = \bigvee_i \bigwedge_j \varphi_{ij}[j]$. We let i vary over $\{1, \ldots, n\}$ and set $J_i = \{F \in TR^*(\widetilde{\Sigma}) \mid F \models \bigwedge_{j \in \mathcal{P}} \varphi_{ij}[j]\}$. Note that the models relation in the definition of J_i is that of $\mathrm{DFO}(\widetilde{\Sigma})$. It is clear that $J = \bigcup_i J_i$ and hence $\lim(J) = \bigcup_i \lim(J_i)$. Now we show that $\lim(J_i) \cap TR_A$ is $\mathrm{DLFO}(\widetilde{\Sigma})$-definable. Note that J_i is also a $\mathcal{P}_A$-pointed $\mathrm{FO}(\widetilde{\Sigma})$-definable trace language of $TR^*(\widetilde{\Sigma})$.

In order to keep the notation clean, we rename J_i by M. More precisely, we consider M to be a $\mathcal{P}_A$-pointed $\mathrm{FO}(\widetilde{\Sigma})$-definable trace language of $TR^*(\widetilde{\Sigma})$ with $M = L^*_{\vartheta}$ where ϑ (a sentence of $\mathrm{FO}(\widetilde{\Sigma})$) admits a simple first normal form: $\vartheta = \bigwedge_{j \in \mathcal{P}} \vartheta_j[j]$ where, for each $j \in \mathcal{P}$, ϑ_j is a sentence of $\mathrm{FO}(\widetilde{\Sigma})$.

Let $F = (E, \leq, \lambda) \in TR^*(\widetilde{\Sigma})$. For $j \in \mathcal{P}$, we set $c_j = \downarrow E_j$. Then for $j \in \mathcal{P}$, $F \models \vartheta_j[j]$ iff $F_{c_j} \models \vartheta_j$. Further, $F \in M$ iff $F \models \vartheta$ iff $F \models \vartheta_j[j]$ for all $j \in \mathcal{P}$.

Fix $1 \leq m \leq k$. We can easily construct a sentence ϑ'_{j_m} of $\mathrm{FO}(\widetilde{\Sigma})$ from $\{\vartheta_j\}_{j \in P_m}$ such that the following holds. Let $F = (E, \leq, \lambda) \in TR^*(\widetilde{\Sigma})$, $c = \downarrow E_{j_m}$ and for each $j \in P_m$, $c_j = \downarrow^j(c)$. Then $F_c \models \vartheta'_{j_m}$ iff for each $j \in P_m$, $F_{c_j} \models \vartheta_j$.

The next lemma completes the proof of Claim 2 by showing that $\lim(M) \cap TR_A$ is $\mathrm{DLFO}(\widetilde{\Sigma})$-definable. See [AS2] for the details.

Lemma 4. *With* $\vartheta' = \left(\bigwedge_{j \in Q} \lim^*_j(\vartheta_j)\right) \wedge \left(\bigwedge_{1 \leq m \leq k} \lim^{\omega}_{j_m}(\vartheta'_{j_m})\right) \wedge \varphi_A$, *as a sentence of* $\mathrm{DLFO}(\widetilde{\Sigma})$, $L_{\vartheta'} = \lim(M) \cap TR_A$.

4 Normal Forms for Monadic Second-Order Logic

This section presents the normal forms for the monadic second-order theory $\mathrm{MSO}(\widetilde{\Sigma})$ over traces. $\mathrm{MSO}(\widetilde{\Sigma})$ extends $\mathrm{FO}(\widetilde{\Sigma})$ by second-order variables which range over sets of events in traces. We skip the discussion of formulas, interpretations and models of $\mathrm{MSO}(\widetilde{\Sigma})$ which is completely straightforward. The notable difference is a new atomic formula $x \in S$ to test if a first-order variable x is a member of a second-order variable S with the obvious interpretation. Following ideas in Section 1, it is clear how to define the local theories $\{\mathrm{MSO}_i(\widetilde{\Sigma})\}_{i \in \mathcal{P}}$ and the distributed monadic second-order theory $\mathrm{DMSO}(\widetilde{\Sigma})$.

The first normal form asserts that $\mathrm{MSO}(\widetilde{\Sigma})$ is equivalent to $\mathrm{DMSO}(\widetilde{\Sigma})$. It is easy to formulate this in terms of a monadic Ehrenfeucht-Fraïssé game where sets of elements are chosen in some of the moves of a play apart from individual elements in the remaining moves. Finally, we are led to a construction of a winning strategy for a "global" monadic EF-game from the winning strategies for "local" monadic EF-games. This can be done almost verbatim as in the proof of Claim 1. Here we provide the main ingredients. The initial rounds of the local games are played in the same manner. In the later rounds, Spoiler's choice of a set S in the global game is distributed to multiple local games. Write $S = \cup_{i \in \mathcal{P}} S_i$ as a disjoint union in the lexicographic order. Spoiler makes an additional move in the ith local game by choosing the set S_i which Duplicator responds with R_i. With $R = \cup_{i \in \mathcal{P}} R_i$, Duplicator responds in the global game with R.

It remains to show that atomic formulas cannot distinguish between the two structures on the basis of elements and sets picked during the moves of a play. For the atomic formulas in the first-order theory, we can follow the proof of Claim 1. For the new atomic formula, this follows easily since we are using winning strategies for the local games.

Interestingly, the following strong form of Proposition 1 holds for the monadic second-order theory. As seen before, it does *not* hold for the first-order theory.

Proposition 3. *There exists a positive integer κ depending only on $\widetilde{\Sigma}$ such that, for $n \geq 0$, $Q \subseteq \mathcal{P}$ and $F \in TR(\widetilde{\Sigma})$, the n-theory $\mathrm{MTh}_n(F_Q)$ is determined by the collection $\{\mathrm{MTh}_{n+\kappa}(F_i)\}_{i \in Q}$ of $n{+}\kappa$-theories.*

The m in MTh_m, for simplicity, counts both the first-order and monadic second-order quantifications. The proof follows essentially from the proof of Claim 1 by invoking the next lemma (Lemma 5) instead of Lemma 2.

Lemma 5. *There exists an integer κ with the following property. Fix $m \geq \kappa$ and $i, j \in \mathcal{P}$. Let $F, H \in TR^*(\widetilde{\Sigma})$ such that $\mathrm{MTh}_m(F_i) = \mathrm{MTh}_m(H_i)$ and $\mathrm{MTh}_m(F_j) = \mathrm{MTh}_m(H_j)$. Then, for all $k, l \in \mathcal{P}$, $\mathsf{latest}_{i \to k}(F) = \mathsf{latest}_{j \to l \to k}(F)$ iff $\mathsf{latest}_{i \to k}(H) = \mathsf{latest}_{j \to l \to k}(H)$.*

We give a brief sketch of the proof. The proof relies on the crucial result in [MS] that the latest information can be kept track of by a deterministic asynchronous automaton [Zie] whose size depends only on $\widetilde{\Sigma}$. We refer to this automaton as the *gossip automaton.* Let $i, j \in \mathcal{P}$. Then it is possible to compare the latest information of i and j in a trace F by looking *only* at the states of i and j assumed at the end of the unique run of the gossip automaton on F. Thanks to the asynchronous nature of the gossip automaton, these states depend only on F_i and F_j respectively. For each state s of i, we associate a sentence φ_s of $\mathrm{MSO}(\widetilde{\Sigma})$ such that, for $F \in TR(\widetilde{\Sigma})$, $F_i \models \varphi_s$ iff i assumes state s at the end of the unique run of the gossip automaton on F. The construction of φ_s involves an encoding of the gossip automaton (see [Tho1]). Now it is routine and straightforward to complete the proof.

Following Section 3, it is immediate how to define the distributed limit monadic second-order theory $\mathrm{DLMSO}(\widetilde{\Sigma})$. Recall that the intermediate normal form also holds for $\mathrm{MSO}(\widetilde{\Sigma})$-definable trace languages. This normal form and the first normal form give the second normal form using techniques in Section 3.

5 Discussion

We show that, as far as expressivity is concerned, the local and global paradigms of reasoning about distributed behaviours are the same. While global logics seem more natural for specification, local logics appear closer to the "implementation" of these specifications. Thus transiting from global specification to local requirements seems to be an important step in the synthesis of distributed systems.

The first normal form provides a direct translation of a global formula into a boolean combination of local formulas. An appealing feature of this translation scheme is that there is only a *constant* blow-up in the quantifier depth for both $\mathrm{FO}(\widetilde{\Sigma})$ and $\mathrm{MSO}(\widetilde{\Sigma})$. Interestingly, the normal form for $\mathrm{MSO}(\widetilde{\Sigma})$ also follows from automata-theoretic characterizations of $\mathrm{MSO}(\widetilde{\Sigma})$-definable trace languages (see [Tho1]) but with non-elementary blow-up in the quantifier depth.

McNaughton's theorem and its analogue in the first-order framework play a very crucial role in the classification of reactive properties of distributed systems such as safety, liveness. The second normal form provides a new classification scheme. A reactive property is said to be an i-liveness property if it can be expressed in the form $\lim_i^\omega(\varphi)$. Other properties such as i-safety may be similarly classified. Thus, for example, in a typical client-server scenario, the server ensures the safety of the system while the clients strive for liveness.

Another important application of the normal forms was in designing new linear time temporal logics over traces [AS1]. An attractive feature of LTL is that it is as expressive as FO and at the same time tractable, unlike FO. However, the natural extension of LTL to traces, though as expressive as FO [DG], is intractable [Wal] and thus there is the question of a tractable temporal logic over traces which equals FO in expressivity [MT]. In [AS1], we have proposed new tractable local linear time temporal logics over traces which are expressively complete. This result crucially uses the normal forms proved in this paper.

Acknowledgement

We thank K. Narayan Kumar and Madhavan Mukund for many discussions. The first author also thanks Prof. W. Thomas and Christof Löding for insights into the connections between logics, automata and languages.

References

AS1. B. Adsul and M. Sohoni: Complete and Tractable Local Linear Time Temporal Logics over Traces, *Proc. ICALP '02, LNCS* **2380** (2002) 926–937.

AS2. B. Adsul and M. Sohoni: First-Order Logic over Traces, Technical Report, Dept. of Computer Science & Engineering, I. I. T. Mumbai, India (2002). Also see `http://www.cse.iitb.ac.in/~abharat/logics.html`

DG. V. Diekert and P. Gastin: LTL is Expressively Complete for Mazurkiewicz Traces, *Proc. ICALP '00, LNCS* **1853** (2000) 211–222.

DM. V. Diekert and A. Muscholl: Deterministic Asynchronous Automata for Infinite Traces, *Acta Inf.*, **31** (1993) 379–397.

DR. V. Diekert and G. Rozenberg (Eds.): *The Book of Traces*, World Scientific, Singapore (1995).

EF. H.-D. Ebbinghaus and J. Flum: *Finite Model Theory*, Springer-Verlag, New York (1995).

EM. W. Ebinger and A. Muscholl: Logical Definability on Infinite Traces, *Proc. ICALP '93, LNCS* **700** (1993) 335–346.

MS. M. Mukund and M. Sohoni: Keeping Track of the Latest Gossip in a Distributed System, *Distributed Computing*, **10**(3) (1997) 137–148.

MT. M. Mukund and P.S. Thiagarajan: Linear Time Temporal Logics over Mazurkiewicz Traces, *Proc. MFCS'96, LNCS* **1113** (1996) 62–92.

Tho1. W. Thomas: On Logical Definability of Trace Languages, *Proc. an ASMICS workshop*, Report TUM-I9002, Technical Univ. of Munich, (1989) 172–182.

Tho2. W. Thomas: Automata on Infinite Objects, in: *Handbook of Theoretical Computer Science Vol. B*, Elsevier, Amsterdam (1990) 135–191.

Wal. I. Walukiewicz: Difficult Configurations - On the Complexity of LTrL, *Proc. ICALP '98, LNCS* **1443** (1998) 140–151.

Zie. W. Zielonka: Notes on Finite Asynchronous Automata, *RAIRO Inform. Théor. Appl.* **21** (1987) 99–135.

On the Hardness of Constructing Minimal 2-Connected Spanning Subgraphs in Complete Graphs with Sharpened Triangle Inequality*
(Extended Abstract)

Hans-Joachim Böckenhauer[1], Dirk Bongartz[1], Juraj Hromkovič[1], Ralf Klasing[2], Guido Proietti[3], Sebastian Seibert[1], and Walter Unger[1]

[1] Lehrstuhl für Informatik I, RWTH Aachen, 52074 Aachen, Germany
{hjb,bongartz,jh,seibert,quax}@cs.rwth-aachen.de
[2] Department of Computer Science, King's College London
Strand, London WC2R 2LS, UK
klasing@dcs.kcl.ac.uk
[3] Dipartimento di Informatica, Università di L'Aquila, 67010 L'Aquila, Italy and
Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR, Roma, Italy
proietti@di.univaq.it

Abstract. In this paper we investigate the problem of finding a 2-connected spanning subgraph of minimal cost in a complete and weighted graph G. This problem is known to be APX-hard, both for the edge- and for the vertex-connectivity case. Here we prove that the APX-hardness still holds even if one restricts the edge costs to an interval $[1, 1 + \varepsilon]$, for an arbitrary small $\varepsilon > 0$. This result implies the first explicit lower bound on the approximability of the general problems.

On the other hand, if the input graph satisfies the sharpened β-triangle inequality, then a $\left(\frac{2}{3} + \frac{1}{3} \cdot \frac{\beta}{1-\beta}\right)$-approximation algorithm is designed. This ratio tends to 1 with β tending to $\frac{1}{2}$, and it improves the previous known bound of $\frac{3}{2}$, holding for graphs satisfying the triangle inequality, as soon as $\beta < \frac{5}{7}$.

Furthermore, a generalized problem of increasing to 2 the edge-connectivity of any spanning subgraph of G by means of a set of edges of minimum cost is considered. This problem is known to admit a 2-approximation algorithm. Here we show that whenever the input graph satisfies the sharpened β-triangle inequality with $\beta < \frac{2}{3}$, then this ratio can be improved to $\frac{\beta}{1-\beta}$.

Keywords: Approximation algorithm, inapproximability, minimum-cost biconnected spanning subgraph, augmentation

* This work was partially supported by DFG-grant Hr 14/5-1, the CNR-Agenzia 2000 Program, under Grants No. CNRC00CAB8 and CNRG003EF8, and the Research Project REAL-WINE, partially funded by the Italian Ministry of Education, University and Research.

1 Introduction

The design of approximation algorithms is one of the most successful approaches in attacking hard optimization problems. Another fundamental approach is trying to specify the set of "easy" input instances, for which the hard problem considered can be solved efficiently. Here we follow the idea of the concept of stability of approximation [6,20] that combines these two approaches. The main point of the stability concept is to try to split the set of all input instances into (potentially infinitely many) classes with respect to the achievable approximation ratio. This approach is similar to the concept of parameterized complexity introduced by Downey and Fellows [11,12]. (The difference is in that we relate the parameter to the approximation ratio while Downey and Fellows relate the parameter to the time complexity.) The crucial point in applying this concept is to find parameters that well capture the difficulty of problem instances of the particular hard problem. The concept of the stability of approximation was successfully applied [6,7] for TSP by using the β-triangle inequality to partition the set of all input instances of TSP. We say, for every $\beta \geqslant \frac{1}{2}$, that a complete weighted graph $G = (V, E)$ satisfies the β-triangle inequality, if, for a given non-negative cost function cost : $E \to \mathbb{R}_0^+$, and for all vertices $u, v, w \in V$, we have that

$$\mathrm{cost}(\{u,w\}) \leqslant \beta\,(\mathrm{cost}(\{u,v\}) + \mathrm{cost}(\{v,w\}))\,.$$

We also speak of a sharpened triangle inequality in the case $\beta < 1$, and of a relaxed triangle inequality in the case $\beta > 1$.

Notice that for $\beta = \frac{1}{2}$ all edges have the same weight, and that the general TSP does not admit any polynomial approximation ratio in the number of vertices of G. In a sequence of papers [1,2,4,5,6,7,8], lower and upper bounds on the polynomial-time approximability of TSP input instances satisfying β-triangle inequalities were proved. These lower and upper bounds continuously grow with β from 1 (for $\beta = \frac{1}{2}$) to infinity (for β tending to infinity) and so the strength of the triangle inequality really captures the hardness of TSP input instances. Very recently, also the approximability of input instances of the asymmetric TSP satisfying a sharpened triangle inequality was investigated in [10].

Although at first glance the sharpened triangle inequality might seem less natural to consider than the relaxed triangle inequality, there are some good reasons to do so. For example, any Euclidean instance, in which the cost function is given by the Euclidean distance and where no three points lie on the same line, satisfies a sharpened triangle inequality with

$$\beta = \max_{u,v,w \in V} \frac{\mathrm{cost}(\{u,w\})}{\mathrm{cost}(\{u,v\}) + \mathrm{cost}(\{v,w\})} < 1.$$

A further discussion on the motivation of the sharpened triangle inequality is given in [6].

The goal of this paper is to extend this characterization to two fundamental connectivity problems, that is the *minimum-cost 2-connected spanning subgraph problem* (note that this may be either 2-edge- or 2-vertex-connectivity) and the *minimum-cost 2-edge-connectivity augmentation problem*. Both problems play a crucial role in graph theory, and have many important applications in the framework of network design, especially with the increasing attention towards communication reliability aspects [3,17,19].

Concerning the former problem, in its more general formulation, we are given a weighted graph $G = (V, E)$, and we want to find a minimum-cost 2-connected spanning subgraph $H = (V, E')$ of G (recall that a graph is 2-edge-connected [2-vertex-connected] if the removal of any edge [vertex] leaves the graph connected). This problem is well-known to be NP-hard [17]. Concerning approximability results, for both kinds of connectivity the best known ratios is 2 [23,25]. This ratio can be lowered to 3/2 if G satisfies the triangle inequality [15], and further decreased to 4/3 for the special case in which G is unweighted [26]. If G is *complete* and Euclidean in $\mathbb{R}^d$, for any integer $d > 0$, (i.e., G is embedded in the Euclidean d-dimensional space and the edge weights correspond to the Euclidean d-dimensional distance between the corresponding endvertices), then the problem admits a PTAS [9]. Finally, concerning inapproximability results, the problem is known to be APX-hard, even if G is unweighted [14] and has bounded degree [9], or is complete and Euclidean in $\mathbb{R}^{\log n}$ [9].

Regarding the latter problem, in its more general formulation, we are given a weighted graph $G = (V, E)$ and a spanning subgraph $H = (V, E')$ of G, and we want to find a minimum-cost set of edges in $E - E'$ whose addition to H augments its edge-connectivity to 2. This problem is well-known to be NP-hard [17], but it becomes polynomial-time solvable if G is complete and unweighted [13]. Another interesting polynomial case arises when H is restricted to be a Hamiltonian path of G [16]. Concerning approximability results, the best known approximation ratio is 2 [22]; if G is unweighted, the ratio can can be improved to $(51/26 + \varepsilon)$, for any constant $\varepsilon > 0$ [24]. Finally, as far as inapproximability results are concerned, these are the same holding for the previous problem.

In this paper, we focus on input graphs which are complete and satisfy a sharpened triangle inequality, and we provide the following three main contributions for the above problems:

(i) We show that the minimum-cost 2-connectivity problems are APX-hard even for the subsets of input instances satisfying the β-triangle inequality for an arbitrary $\beta > \frac{1}{2}$, and even for input instances with costs from an interval $[1, 1 + \varepsilon]$ for an arbitrary small ε. Since we provide an explicit lower bound δ_β on the polynomial-time approximability that grows with β, we obtain the first explicit lower bound on the approximability of 2-connectivity problems and of their generalized augmentation versions.

(ii) We design a $\left(\frac{2}{3} + \frac{1}{3} \cdot \frac{\beta}{1-\beta}\right)$-approximation algorithm for the minimum-cost 2-edge- [2-vertex]-connectivity problem restricted to the input instances satisfying the β-triangle inequality for $\frac{1}{2} < \beta < 1$. This improves, for $\beta < \frac{5}{7}$, the best known approximation ratio of 3/2 holding for instances satisfying the triangle inequality [15].

(iii) We design a $\frac{\beta}{1-\beta}$-approximation algorithm for the input instances satisfying the β-triangle inequality ($\frac{1}{2} < \beta < 1$) of the augmentation problem. This improves, for $\beta < \frac{2}{3}$, the best known approximation ratio of 2 [22].

This paper is organized as follows. Section 2 is devoted to definitions and some useful fundamental observations about input instances satisfying the sharpened triangle inequality. The subsequent three sections present the above mentioned results. Because of the restricted size some proofs are omitted in this extended abstract.

2 Preliminaries

In this section we give details on the notion of the sharpened triangle inequality and some of its implications. The proofs of this section are omitted in this extended abstract.

Lemma 1. *[5] Let $G = (V, E)$ be a weighted complete graph with a cost function* cost*, and $\frac{1}{2} < \beta < 1$. Let $c_{\min}$ and $c_{\max}$ denote the minimal and the maximal cost occurring in G respectively. If the cost function* cost *obeys the β-triangle inequality, then:*

(a) For all adjacent edges $e_1, e_2 \in E$ the inequality $\mathrm{cost}(e_1) \leqslant \frac{\beta}{1-\beta}\mathrm{cost}(e_2)$ *holds.*
(b) $c_{\max} \leqslant \frac{2\beta^2}{1-\beta} c_{\min}$. □

Now we can derive the following:

Corollary 1. *Let $G = (V, E)$ be a weighted complete graph with cost function* cost *that satisfies the β-triangle inequality for $\frac{1}{2} < \beta < 1$. If $\beta < \frac{2}{3}$, then* $\mathrm{cost}(e_3) < \mathrm{cost}(e_1) + \mathrm{cost}(e_2)$ *for all edges $e_1, e_2, e_3 \in E$, where both e_1 and e_2 are adjacent to e_3.*

This result implies that in the case where $\beta < \frac{2}{3}$ holds, we can replace two edges by one adjacent edge, thus decreasing the cost.

Next we formulate the tasks which will be investigated in the rest of the paper. Recall that a graph is said to be *2-edge-connected* (2-ec, for short) if the removal of any edge leaves the graph connected. Similarly, a graph is said to be *2-vertex-connected* (2-vc, for short) if the removal of any vertex leaves the graph connected. Notice that every 2-vc graph is also a 2-ec graph.

Definition 1. *Let $G = (V, E)$ be a complete, weighted graph. The* 2-ec spanning subgraph (2-ECSS) problem *is that of computing a minimum weight spanning 2-ec subgraph $G' = (V, E')$ of G.*

The *2-vc spanning subgraph (2-VCSS) problem* can be defined similarly.

Definition 2. *Let $G = (V, E)$ be a complete, weighted graph, and let $H = (V, E')$ be a spanning subgraph of G. The* 2-ec augmentation (2-ECA) problem *on inputs G and H is that of computing a minimum weight set of edges A such that $G' = (V, E' \cup A)$ is a spanning 2-ec subgraph of G.*

In the following, a *component* [*block*] will denote a 2-ec [2-vc] subgraph which is maximal w.r.t. inclusion. Furthermore, we will use the term *leaf* to denote a vertex of degree 1.

3 A Lower Bound on the Approximability of the 2-ECSS and the 2-VCSS Problem

We show in the following that in the case of the β-triangle inequality with $\frac{1}{2} < \beta < \frac{2}{3}$, the 2-ECSS and the 2-VCSS problems are equivalent to computing a minimum-cost Hamiltonian cycle, i.e., to solving the TSP for graphs with β-triangle inequality. In this way, we directly obtain the APX-hardness of the problem even if the edge costs are restricted to $[1, 1 + \varepsilon]$, for an arbitrary small positive constant ε, as well as the first explicit lower bound on the approximability of the problems [8].

Lemma 2. *Every spanning 2-ec subgraph $G' = (V, E')$ of a complete, weighted graph $G = (V, E)$ satisfying the β-triangle inequality for $\frac{1}{2} < \beta < \frac{2}{3}$, can be transformed into a spanning 2-ec subgraph $G'' = (V, E'')$ of G that has the following properties:*

1. *G'' is a Hamiltonian cycle, and*
2. *either G'' has lower weight than G', or $G'' = G'$.*

Proof. First, recall the fact that in a 2-ec subgraph each vertex has to occur in at least one cycle.

To prove the lemma we now describe transformations which decrease the weight of the 2-ec subgraph and none of which is applicable only if any two cycles of the subgraph share no common vertex. Since the transformations preserve the 2-ec property, the resulting subgraph will be a Hamiltonian cycle. We consider the following cases:

– Let C_1, C_2 be two cycles that have a region where they share exactly one vertex v.
  Determine two edges $e_1 = \{v, w\}, e_2 = \{v, w'\}$ incident to v, where $e_1 \in C_1$ and $e_2 \in C_2$. Replace the edges e_1 and e_2 by the edge $e_3 = \{w, w'\}$. Since e_1, e_2, and e_3 form a triangle, we can directly apply the β-triangle inequality, thus ensuring that the weight of the resulting subgraph has decreased. Obviously the graph remains 2-ec. This procedure is shown in Figure 1 (i).
– Consider two cycles C_1, C_2 having a region where they share exactly one edge $\{v_1, v_2\}$. In this case we can just remove the edge $\{v_1, v_2\}$ from the 2-ec subgraph and obtain a 2-ec subgraph of decreased weight (see Figure 1 (ii)).
– Assume that the cycles C_1 and C_2 have a region where they have a contiguous series of vertices $v_1, \ldots, v_k$ in common, $k \geqslant 3$. In this case we connect the cycles as follows. We determine two edges $e_1 = \{w, v_1\}$ and $e_2 = \{v_k, w'\}$ belonging to the cycles C_1 and C_2, respectively, and replace them by the edge $e_3 = \{w, w'\}$, thus decreasing the weight of the resulting 2-ec subgraph

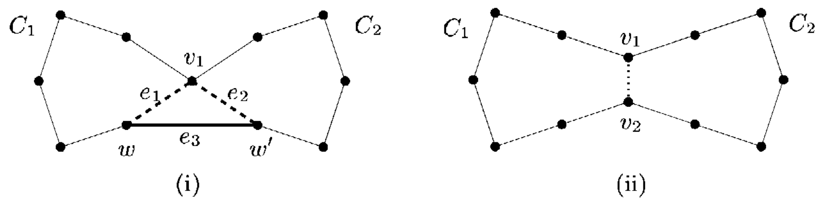


Fig. 1. Strategy to join two cycles having a region where they share exactly one vertex (i) or exactly one edge (ii).

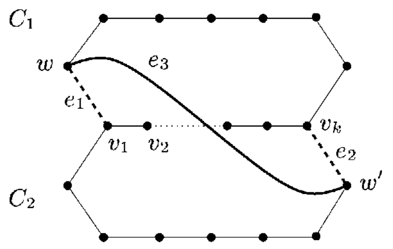


Fig. 2. Strategy to join two cycles having a series of contiguous common vertices $v_1, \ldots, v_k$.

as guaranteed by Corollary 1. We present a schematic view of this strategy in Figure 2.
Note that this transformation also decreases the weight of the graph and the number of edges if the two cycles have some additional vertices ($\notin \{v_1, \ldots, v_k\}$) in common.

To ensure the termination of the transformation process it is not necessary to show that the number of cycles decreases in each step, but it is sufficient to recognize that we reduce the number of edges by exactly one in each transformation step.

We apply these transformations to the given spanning 2-ec subgraph G' until none of them is applicable and denote the resulting graph by G''.

The above transformations guarantee that eventually each vertex in G'' belongs to exactly one cycle. Since we know that G' is a spanning 2-ec subgraph, we also know that G'' is connected, because the transformations do not affect the connectivity. Hence G'' is a Hamiltonian cycle, and this completes the proof. □

Since every Hamiltonian cycle is 2-vc [2-ec], Lemma 2 directly implies the following.

Lemma 3. *Let $G = (V, E)$ be a complete, weighted graph satisfying the β-triangle inequality with $\frac{1}{2} < \beta < \frac{2}{3}$. Then, to compute a minimum-cost 2-ec [2-vc] subgraph $G' = (V, E')$ of G is equivalent to computing a minimum-cost Hamiltonian cycle.* □

Thus, we can transfer all lower as well as upper bounds on the approximability of the TSP with sharpened triangle inequality to the 2-ECSS and the 2-VCSS problems where $\frac{1}{2} < \beta < \frac{2}{3}$. We therefore obtain the following inapproximability result:

Theorem 1. *For any $\varepsilon > 0$, it is NP-hard to approximate the 2-ECSS and the 2-VCSS problem on a graph with β-triangle inequality with $\frac{1}{2} < \beta < \frac{2}{3}$ within $\frac{7611+10\beta^2+5\beta}{7612+8\beta^2+4\beta} - \varepsilon$.*

Proof. Direct implication from Lemma 3 and the lower bound on the approximability of the TSP with sharpened β-triangle inequality proved in [8]. □

Note that Theorem 1 directly implies that the 2-ECSS and 2-VCSS problems are APX-hard also for instances where the edge costs are restricted to the interval $[1, 1+\varepsilon]$ for an arbitrary small $\varepsilon > 0$ (see Lemma 1 (b)).

Since the above bound is monotonically increasing, by letting β tend to $\frac{2}{3}$, we obtain the following general result:

Theorem 2. *For any $\varepsilon > 0$, it is NP-hard to approximate the 2-ECSS and the 2-VCSS problem on general weighted graphs within $\frac{68569}{68564} - \varepsilon$.* □

4 An Approximation Algorithm for the 2-ECSS and the 2-VCSS Problem

In this section we will present an approximation algorithm for the 2-ECSS and the 2-VCSS problem that is based on the construction of a minimum-cost spanning subgraph in which each vertex has degree at least two.

Degree-2 Subgraph Algorithm

Input: A complete graph $G = (V, E)$ with a cost function cost : $E \to \mathbb{R}^{\geqslant 0}$ satisfying the β-triangle inequality for $\frac{1}{2} < \beta < 1$.
Step 1: Construct a minimum cost spanning subgraph $G' = (V, E')$ of G in which each vertex has degree at least two. (Note that G' is not necessarily connected.)
Step 2: Find the components of G' containing at least three vertices. Denote the set of these components by $C = \{C_1, \ldots, C_k\}$. (Note that each component contains either exactly one or at least three vertices.)
Step 3: For $1 \leqslant i \leqslant k$, find the cheapest edge $\{a_i, b_i\}$ in every component C_i of C[1].
Step 4: Obtain a 2-ec spanning subgraph H of G from G' by replacing the edges $\{\{a_i, b_i\} \mid 1 \leqslant i \leqslant k\}$ by the edges $\{\{b_i, a_{i+1}\} \mid 1 \leqslant i \leqslant k-1\} \cup \{\{b_k, a_1\}\}$.
Step 5: Set $H' = H$. Find the blocks of H'. While a cutvertex v (i.e., a vertex whose removal from H' along with the removal of its incident edges disconnects H') remains, find edges $\{u, v\}$ and $\{v, w\}$ in different blocks and replace $\{u, v\}$ and $\{v, w\}$ by $\{u, w\}$.
Output: The 2-ec spanning subgraph H and the 2-vc spanning subgraph H' of G.

[1] Since at most one vertex of C_i is adjacent to C_{i+1}, we choose the names a_i, b_i so that b_i is not adjacent to C_{i+1}.

In the following, we analyze the approximation ratio of the above algorithm, which improves, for $\beta < \frac{5}{7}$, the best known approximation ratio of $\frac{3}{2}$, holding for instances satisfying the triangle inequality [15].

Theorem 3. *For $\frac{1}{2} < \beta < 1$, the above algorithm is a $\left(\frac{2}{3} + \frac{1}{3} \cdot \frac{\beta}{1-\beta}\right)$-approximation algorithm for the 2-ECSS [2-VCSS] problem on a complete graph satisfying the β-triangle inequality.*

Proof. Obviously, the cost of the minimum cost subgraph in which each vertex has degree at least two is a lower bound on the cost of an optimal 2-ec [2-vc] spanning subgraph[2].

Such a subgraph can be found in polynomial time. In fact, the following optimization problem is solvable in polynomial time [18, page 259]: Maximize the sum of edge costs of a subgraph such that the degree of every vertex lies between a given lower bound c and a given upper bound d. If we define $c = 0$ and $d = |V| - 3$, then the edges not belonging to such a subgraph form a subgraph of minimum cost, such that the degree of every vertex is at least 2 (since the degree of every vertex in the complete graph is $|V| - 1$).

Now we will show that the subgraph H constructed in Step 4 is 2-ec. Note that there are no 2-ec graphs with exactly two vertices. Thus, the constructed graph G' has only components with either exactly one or at least three vertices. The components of G' are connected in a forest-like way. More formally speaking, let $\tilde{G}'$ be the forest obtained from G' by shrinking every component of G' into a single vertex. The leaves of $\tilde{G}'$ correspond to components with at least three vertices, since we have assured in Step 1 that every vertex in G' has degree at least two. This implies that every component in G' of size one is connected to at least two of the components in C via edge-disjoint paths.

Now we are ready to prove that the deletion of an arbitrary edge in the graph H constructed in Step 4 does not disconnect H (i.e., H is 2-ec). Note that H is connected even in the case that G' was disconnected, due to the edges added in Step 4. For this proof we will distinguish three cases:

1. First, we will show that the deletion of any edge inside a component C_i, $1 \leqslant i \leqslant k$, does not disconnect H. In the transformation of Step 4, the edge $\{a_i, b_i\}$ in C_i is replaced by a path of edges outside C_i. Since C_i was 2-ec in G', the removal of any of the remaining edges of C_i does not disconnect C_i, and it clearly does not affect the connectivity of the rest of H.
2. The deletion of any of the edges added in Step 4 does not disconnect H, since these edges connect the components from C in the form of a cycle.
3. It remains to show that the deletion of an arbitrary forest edge connecting the components in G' does not disconnect H. This follows, since by the construction, the components of size at least three are connected in H by a cycle,

[2] Note that, for $\beta > 2/3$, the cost of a minimum cost cycle cover is not necessarily a lower bound on the cost of a minimum 2-ec spanning subgraph. For a minimum spanning 2-ec subgraph it only holds that each vertex has degree at least two. This allows the use of one vertex in more than one cycle which can reduce the cost compared to a cycle cover.

and every component of size one is connected to at least two components in C via edge-disjoint paths.

Let us now analyze the cost of H, as compared to the optimum. For every component C_i, $1 \leqslant i \leqslant k$, the cheapest edge is removed from C_i. Thus, the cost of these edges is at most $\frac{1}{3} \cdot \text{cost}(C_i)$ since each of these components has at least three vertices. The removed edges are replaced by adjacent edges, and according to Lemma 1 the costs of these new edges can exceed the costs of the removed edges by a factor of at most $\frac{\beta}{1-\beta}$. Thus, we have

$$\text{cost}(H) \leqslant \left(\frac{2}{3} + \frac{1}{3} \cdot \frac{\beta}{1-\beta}\right) \cdot \text{cost}(G').$$

It remains to show that the subgraph H' constructed in Step 5 is 2-vc and at least as cheap as the 2-ec subgraph H. We know that each cutvertex is incident to at least four edges, since a cutvertex connects to at least two blocks, and thus the existence of a cutvertex with at most three edges would imply the existence of a bridge in H which is a contradiction to the fact that H is 2-ec. Thus, we can apply the transformation in Step 5 to every cutvertex of H, and hence the resulting subgraph H' is 2-vc. Since the edges $\{u, v\}$, $\{v, w\}$ and $\{u, w\}$ form a triangle, the resulting graph must become cheaper in each transformation step, due to the β-triangle inequality. □

Note that the above analysis of Step 5 of the algorithm directly implies that for graphs obeying the β-triangle inequality for $\frac{1}{2} < \beta < 1$, a minimum-cost 2-ec subgraph is also a minimum-cost 2-vc subgraph and vice versa.

5 An Approximation Algorithm for the 2-ECA Problem

In the sequel we will present an approximation algorithm for the 2-ECA problem for complete graphs satisfying the β-triangle inequality for $\frac{1}{2} < \beta < 1$.

As a first step towards this result, we present an approximation algorithm where the input subgraph to be augmented is restricted to be a spanning forest. The idea of this algorithm is to construct a 2-ec subgraph by adding a minimum number of edges. In a first step we will construct a spanning tree by joining the trees of the given forest, and in a second step we will construct a 2-ec subgraph by adding one edge for each two leaves of the spanning tree.

After that we will extend this idea to prove the same approximation ratio for the problem of augmenting arbitrary spanning subgraphs.

Forest-Augmenting Algorithm

Input: A complete graph $G = (V, E)$ with a cost function cost : $E \to \mathbb{R}^{>0}$ satisfying the β-triangle inequality for $\frac{1}{2} < \beta < 1$, and a spanning forest $F = (V, E_F)$ of G.

Step 1: Construct a spanning tree $T = (V, E_T)$ of G with $E_F \subseteq E_T$ by connecting the trees in F in the following way:

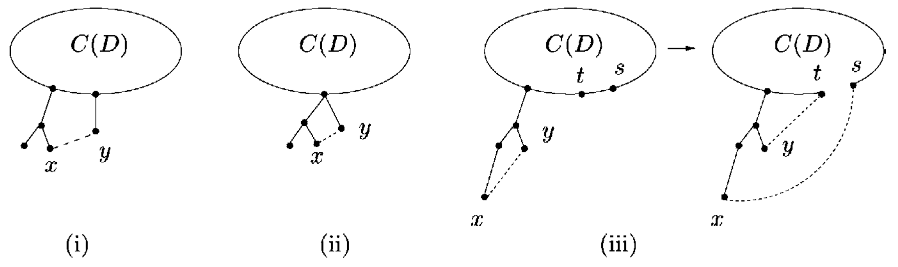


Fig. 3. Some examples for the transformations in Step 2.3 of the Forest-Augmenting Algorithm. In (i) and (ii) possible transformations are shown for the case that the cycle in $D \cup \{x, y\}$ containing $\{x, y\}$ shares at least one common vertex with $C(D)$, and a possible transformation for the other case is presented in (iii).

```
    1.1 Connect the isolated vertices in $F$ by an arbitrary path.
    1.2 Connect the resulting trees by a set of pairwise non-adjacent edges, each of
        which connects two leaves of different trees.
Step 2:
    2.1 Find two leaves $u$ and $v$ in $T$.
    2.2 Set $E_D := E_T \cup \{u, v\}$, set $D := (V, E_D)$. Let $C(D)$ denote the component
        of $D$ containing the vertices $u$ and $v$. (Note that $C(D)$ is a cycle after the
        execution of Step 2.2.)
    2.3 while $D$ contains at least two leaves $x, y$ do
            if there exists a cycle in $D \cup \{x, y\}$ containing $\{x, y\}$ and sharing a
                common vertex with $C(D)$,
            then set $E_D := E_D \cup \{x, y\}$, (see Figure 3 (i),(ii))
            else find an edge $\{s, t\}$ in $C(D) - T$, and set $E_D := (E_D - \{s, t\}) \cup$
                $\{x, s\} \cup \{y, t\}$ (see Figure 3 (iii)).
Step 3: If $D$ contains exactly one leaf $x$ which is adjacent to a vertex $y$ in $D$, let
    $E_D := D \cup \{x, z\}$, where $z \notin \{x, y\}$ is an arbitrary vertex that is neither a leaf
    nor an isolated vertex in $F$ if such a vertex $z$ exists. Otherwise, let $z \notin \{x, y\}$
    be a vertex such that $\text{cost}(\{x, z\}) = \min\{\text{cost}(\{x, r\}) \mid r \neq y\}$.
Output: The 2-ec subgraph $D$ of $G$.
```

The proofs of the following technical lemmas are omitted in this extended abstract.

Lemma 4. *Let $G = (V, E)$ be a complete graph, and let $F = (V, E_F)$ be a spanning forest of G. Every 2-ec subgraph of G containing F has at least as many edges as the subgraph constructed by the algorithm.* □

Lemma 5. *The subgraph D computed by the algorithm provides a solution to the 2-ECA problem on inputs G and F.* □

Theorem 4. *Let $G = (V, E)$ be a complete graph with a cost function* cost : $E \to \mathbb{R}^{>0}$ *satisfying the β-triangle inequality for $\frac{1}{2} < \beta < 1$, and let F be a spanning forest of G. Then, the above algorithm is a $\frac{\beta}{1-\beta}$-approximation algorithm for the 2-ECA problem on inputs G and F.*

We also have to omit the full proof of Theorem 4 due to space limitations, but we give the underlying idea of the proof here. We will estimate the costs of the augmenting edges chosen by the algorithm in terms of the costs of the augmenting edges in an optimal solution. This comparison is done by constructing a set of disjoint paths, with edges alternatingly taken from the constructed and the optimal augmentation, such that each path begins with an edge from the constructed solution and ends with an edge from the optimal solution. By estimating the costs of any edge of the constructed solution in terms of the following edge on the path, we obtain the claimed approximation ratio of $\frac{\beta}{1-\beta}$, according to Lemma 1.

Next we extend the above result to the general case where we have to augment an arbitrary spanning subgraph. The idea of this extension is to shrink the components of the given subgraph to single vertices, and then to apply a modification of the Forest-Augmenting Algorithm. In this way, we improve, for $\beta < \frac{2}{3}$, the best known approximation ratio of 2 [22]. The proof of the theorem is omitted in this extended abstract.

Theorem 5. *Let $G = (V, E)$ be a complete graph with a cost function* cost : $E \to \mathbb{R}^{>0}$ *satisfying the β-triangle inequality for $\frac{1}{2} < \beta < 1$, and let H be a spanning subgraph of G. Then there exists a polynomial time $\frac{\beta}{1-\beta}$-approximation algorithm solving the 2-ECA problem on inputs G and H.* $\square$

Acknowledgment

We thank Eva Trenklerová for helpful comments on a previous version of this paper.

References

1. T. Andreae: On the traveling salesman problem restricted to inputs satisfying a relaxed triangle inequality. *Networks* 38(2) (2001), pp. 59–67.
2. T. Andreae, H.-J. Bandelt: Performance guarantees for approximation algorithms depending on parametrized triangle inequalities. *SIAM Journal on Discrete Mathematics* 8 (1995), pp. 1–16.
3. G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, M. Protasi: *Complexity and Approximation: Combinatorial Optimization Problems and their Approximability Properties.* Springer 1999.
4. M. A. Bender, C. Chekuri: Performance guarantees for the TSP with a parameterized triangle inequality. *Information Processing Letters* 73 (2000), pp. 17–21.
5. H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, W. Unger: Approximation algorithms for the TSP with sharpened triangle inequality. *Information Processing Letters* 75, 2000, pp. 133–138.

6. H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, W. Unger: Towards the Notion of Stability of Approximation for Hard Optimization Tasks and the Traveling Salesman Problem (Extended Abstract). Proc. *CIAC 2000*, LNCS 1767, Springer 2000, pp. 72–86. Full version in *Theoretical Computer Science* 285(1) (2002), pp. 3–24.
7. H.-J. Böckenhauer, J. Hromkovič, R. Klasing, S. Seibert, W. Unger: An Improved Lower Bound on the Approximability of Metric TSP and Approximation Algorithms for the TSP with Sharpened Triangle Inequality (Extended Abstract). Proc. *STACS 2000*, LNCS 1770, Springer 2000, pp. 382–394.
8. H.-J. Böckenhauer, S. Seibert: Improved lower bounds on the approximability of the traveling salesman problem. *RAIRO - Theoretical Informatics and Applications* 34, 2000, pp. 213–255.
9. A. Czumaj, A. Lingas: On approximability of the minimum-cost k-connected spanning subgraph problem. Proc. *SODA'99*, 1999, pp. 281–290.
10. L. S. Chandran, L. S. Ram: Approximations for ATSP with Parametrized Triangle Inequality. Proc. *STACS 2002*, LNCS 2285, Springer 2002, pp. 227–237.
11. R. G. Downey, M. R. Fellows: Fixed-parameter tractability and completeness. *Congressus Numerantium* 87 (1992), pp. 161–187.
12. R. G. Downey, M. R. Fellows: *Parameterized Complexity.* Springer 1999.
13. K. P. Eswaran, R. E. Tarjan: Augmentation Problems. *SIAM Journal on Computing* 5(4), 1976, pp. 653–665.
14. C. G. Fernandes: A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *J. Algorithms*, 28(1) (1998), pp. 105–124.
15. G. N. Frederickson, J. Jájá: On the relationship between the biconnectivity augmentation and traveling salesman problems. *Theoretical Computer Science*, 19 (1982), pp. 189–201.
16. A. Galluccio, G. Proietti: Polynomial time algorithms for edge-connectivity augmentation of Hamiltonian paths. Proc. *ISAAC'01*, LNCS 2223, Springer 2001, pp. 345–354.
17. M. R. Garey, D. S. Johnson: *Computers and Intractability: A guide to the theory of NP-completeness.* W. H. Freeman and Company, San Francisco, 1979.
18. M. Grötschel, L. Lovász, A. Schrijver: *Geometric Algorithms and Combinatorial Optimization.* Springer 1988.
19. M. Grötschel, C.L. Monma, M. Stoer: Design of survivable networks. *Handbooks in OR and MS, Vol. 7*, Elsevier (1995), pp. 617–672.
20. J. Hromkovič: *Algorithmics for Hard Problems - Introduction to Combinatorial Optimization, Randomization, Approximation, and Heuristics.* Springer 2001.
21. P. Krysta, V. S. A. Kumar: Approximation algorithms for minimum size 2-connectivity problems. Proc. *STACS 2001*, LNCS 2010, Springer 2001, pp. 431–442.
22. S. Khuller, R. Thurimella: Approximation algorithms for graph augmentation. *Journal of Algorithms* 14 (1993), pp. 214–225.
23. S. Khuller, U. Vishkin: Biconnectivity approximations and graph carvings. *Journal of the ACM* 41 (1994), pp. 214–235.
24. H. Nagamochi, T. Ibaraki: An approximation for finding a smallest 2-edge-connected subgraph containing a specified spanning tree. Proc. *COCOON'99*, LNCS 1627, Springer 1999, pp. 31–40.
25. M. Penn, H. Shasha-Krupnik: Improved approximation algorithms for weighted 2- and 3-vertex connectivity augmentation. *Journal of Algorithms* 22 (1997), pp. 187–196.
26. S. Vempala, A. Vetta: Factor 4/3 approximations for minimum 2-connected subgraphs. Proc. *APPROX 2000*, LNCS 1913, Springer 2000, pp. 262–273.

Communication Interference in Mobile Boxed Ambients[⋆]

Michele Bugliesi[1], Silvia Crafa[1], Massimo Merro[2], and Vladimiro Sassone[3]

[1] Università "Ca' Foscari", Venezia
[2] École Polytechnique Fédérale de Lausanne
[3] University of Sussex

Abstract. Boxed Ambients (BA) replace Mobile Ambients'open capability with communication primitives acting across ambient boundaries. Expressiveness is achieved at the price of communication interferences on message reception whose resolution requires synchronisation of activities at multiple, distributed locations. We study a variant of BA aimed at controlling communication interferences as well as mobility ones. Our calculus draws inspiration from Safe Ambients (SA) (with passwords) and modifies the communication mechanism of BA. Expressiveness is maintained through a new form of co-capability that at the same time registers incoming agents with the receiver ambient and performs access control.

Introduction

The calculus of Mobile Ambients [3] (MA) introduced the notion of ambient acting at the same time as administrative domain and computational environment. Processes live inside ambients, and inside ambients compute and interact. Ambients relocate themselves, carrying along all their contents: their migration, triggered by the processes they enclose, models mobility of entire domains and active computational loci. Two capabilities control ambient movements: in and out. These are performed by processes wishing their enclosing ambient to move to a sibling and, respectively, out of its parent, as show below, where P, Q and R are processes, m and n ambient names, $\mid$ is parallel composition, and square brackets delimit ambients' contents.

$$n[\,\mathsf{in}\; m.P \mid Q\,] \mid m[\,R\,] \longrightarrow m[\,n[\,P \mid Q\,] \mid R\,],$$
$$m[\,n[\,\mathsf{out}\; m.P \mid Q\,] \mid R\,] \longrightarrow n[\,P \mid Q\,] \mid m[\,R\,]$$

A third capability, open, can be used to dissolve ambients, as expressed by the reduction $\mathsf{open}\; n.P \mid n[\,Q\,] \longrightarrow P \mid Q$. Process interaction is by anonymous message exchanges confined inside ambients, as in

$$n[\,\langle M\rangle.P \mid (x).Q\,] \longrightarrow n[\,P \mid Q\{x := M\}\,],$$

where brackets represent outputs, curly brackets substitutions, and round parentheses bind input variables.

[⋆] Research supported by 'MyThS: Models and Types for Security in Mobile Distributed Systems', EU FET-GC IST-2001-32617.

These ideas have given rise to an innovative calculus capturing several aspects of current real-world distributed systems, and have posed some new hard problems. Paper [7] unveiled a set of so-called grave interferences, i.e. situations where the inherent nondeterminism of movement goes wild. For instance, in

$$k[\,n[\,\mathsf{in}\ m.P \mid \mathsf{out}\ k.R\,] \mid m[\,Q\,]\,]$$

although n's next hop is beyond n's control, the difference that such a choice brings about is so big that it is difficult to see how such a situation could have been purposely programmed. Levi and Sangiorgi's proposal of Safe Ambients (SA) [7] counters it by using 'co-actions' to grant ambients a form of control over other ambients' access. A process willing to be entered will manifest that explicitly, as e.g. in

$$n[\,\mathsf{in}\ m.P \mid Q\,] \mid m[\,\overline{\mathsf{in}}\ m.R \mid S\,] \longrightarrow m[\,n[\,P \mid Q\,] \mid R \mid S\,],$$

and similarly for out and open. Building on such infrastructure, a type-system enforced notion of *single-threadedness* ensures that at any time ambients are willing to engage in at most one activity across boundaries that may lead to grave interferences.

Recently, Merro and Hennessy [8] found useful to work with a version of SA called SAP, where incoming ambients must be able to present a suitable password in order to cross ambients' boundaries. Paper [8] develops a treatable semantic theory for SAP in the form of a labelled transition system (LTS) based characterisation of its (reduction) barbed congruence. We will find use for some of these ideas in the present paper too.

Another source of potential problems is open in its own nature ambient dissolver. A process exercising such a capability will embody all the contents of the dissolved ambient, including its capabilities and migration strategies. Of course there is nothing inherently wrong with that and, as a matter of fact, it is where a large part of the MA's expressiveness resides. For instance, without it the tree structure of a system could never shrink, but only be translated about (or grow). As a consequence, processes would belong to the same ambient for their entire lifespan. Despite its usefulness, from the system designer's point of view open must be handled with the greatest care.

The calculus of Boxed Ambients [2] (BA) was born out of the observation that, after all, there is an alternative way to yield expressiveness. Namely, direct communication across boundaries as in the Seal calculus [15]. As shown below, in BA it is possible to draw an input from a sub-ambient n's local channel (viz. $(x)^n$) as well as from the parent's local channel (viz. $(x)^\uparrow$), and dually with the roles of input and output swapped.

$$\begin{array}{l}(x)^n.P \mid n[\,\langle M\rangle.\,Q \mid R\,] \longrightarrow P\{x := M\} \mid n[\,Q \mid R\,]\\ \langle M\rangle.P \mid n[\,(x)^\uparrow.\,Q \mid R\,] \longrightarrow P \mid n[\,Q\{x := M\} \mid R\,].\end{array}$$

Such design choices, although remarkable in many respects, have the drawback of introducing a great amount of non-local nondeterminism and communication interference. This is exemplified perfectly by the term below, where a single message issued in n unleashes a nondeterministic race among three potential receivers located in three different ambients, possibly belonging to three different administrative domains.

$$m[\,(x)^n.P \mid n[\,\langle M\rangle \mid (x).Q \mid k[\,(x)^\uparrow.R\,]\,]\,]$$

This raises difficulties for a distributed implementation of BA, as there is a hidden, non trivial distributed consensus problem to address at each communication. These forms of interference are as grave as those that led to the definition of SA, as they too should be regarded as programming errors.

In this paper we propose a variant of BA aimed at addressing such issue and at providing a fresh foundation for the ideas behind BA. Our proposal, NBA, takes inspiration from SA(P), and is based on the idea that each ambient comes equipped with two mutually non-interfering channels, respectively for local and upward communications.

$$\begin{array}{l}(x)^n.P \mid n[\langle M\rangle^{\hat{}}.\,Q \mid R\,] \longrightarrow P\{x := M\} \mid n[\,Q \mid R\,]\\ \langle M\rangle^n.P \mid n[(x)^{\hat{}}.\,Q \mid R\,] \longrightarrow P \mid n[\,Q\{x := M\} \mid R\,]\end{array}$$

Hierarchical communication, whose rules are shown above, is indicated by a pair of distinct constructors, simultaneously on input and output, so that no communication interference is possible. Observe that the upward channel can be thought of as a gateway between parent and child, located at the child's and travelling with it, and poses no particular implementation challenges.

From the theoretical viewpoint, a first consequence of the elimination of unwanted interferences is a set of good, expected algebraic laws for NBA, as illustrated in §1.2. Also, the type system for BA results considerably simplified. In particular, the types of ambients and capabilities need only record upward exchanges, while processes are characterised by their local and hierarchical exchanges. The details will be discussed in §2.

Observe however that limiting ourselves to banning communication interferences as above would result in a poorly expressive calculus (although some of its good properties have been underlined in [5]). For instance, in $n[P]$ there would be no way for P to communicate with its sub-ambients, unless their names were statically known. In our effort to tackle interference we seem to have killed hierarchical communication at all. Far from that, as a matter of fact in order to regain expressive power we only need to reinstate a mechanism for an ambient to learn dynamically the names of incoming ambients. Essentially, our idea is to introduce co-actions of the form $\overline{\mathsf{enter}}(x)$ that have the effect of binding such names to the variable x.

Observe that a purely binding mechanism such as this would not in itself be able to control access, but only to registers it. Similarly to SA, it expresses a general willingness to accept incoming ambients; in addition to that, the receiving ambient learns the incoming ambient's name. It can thus be thought as (an abstraction of) an access protocol as it would actually take place in real computational domains, where newly arrived agents would have to register themselves in order to be granted access to local resources. In order to provide ambients with a finer mechanism of access control, we add a second component to our (co-)capabilities and write rules as the one below.

$$a[\,\mathsf{enter}\langle b,k\rangle.P_1 \mid P_2\,] \mid b[\,\overline{\mathsf{enter}}(x,k).Q_1 \mid Q_2\,] \longrightarrow b[\,a[\,P_1 \mid P_2\,] \mid Q_1\{x := a\} \mid Q_2\,]$$

In practical terms, this enhances our access protocol with a form of controlling the credentials of incoming processes (k in the rule above), as a preliminary step to the registration protocol. An example for all of the practical relevance and naturality of this mechanism, is the negotiation of credential that takes place when connecting to a wireless LAN or to a LAN using DHCP or to a ISP using PPP.

Remarkably, our admission mechanism resembles quite closely the notion of passwords as developed in [8], which thus arises yet again as a natural notion to consider. As a consequence, we benefit from results similar to those in *loc. cit.* In particular, we devise a labelled transition semantics (that due to space limitations only appears in the full paper) for NBA that yield a bisimulation congruence sound with respect to (reduction) barbed congruence, and we use it to prove a number of laws. Passwords also have a relevant role in the type system, where their types keep track of the type of (upward exchanges of) incoming ambients, so contributing effectively to a clean and easy type system.

As the paper will show, besides having practical, implementation-oriented features and enjoying good theoretical properties, such as a rich and tractable algebraic theory and a simple type system, at the same time NBA remains expressive enough. In particular, by means of examples and encodings in §3 we show that the expressive power we loose with respect to BA is, as expected and planned, essentially that directly related to communication interferences.

Structure of the Paper. §1 introduces the calculus, presents the reduction semantics and the associated notion of behavioural equivalence, and presents some of its characteristic algebraic laws. The type system of NBA is illustrated and discussed in §2, as §3 focuses on expressiveness issues in relation to BA and SA, including several examples and an encoding of BA into (an extension of) NBA.

1 The NBA Calculus

The syntax of processes is given in the following and is similar to that of BA, except that, as in [8], each of the original capabilities has a co-capability with an extra argument which may be looked upon as a *password*. However, unlike [8], the target ambient of a move gets to know the name of the entering ambient as a result of synchronisation.
Names: a,b,... n,x,y,... $\in \mathbf{N}$

Locations:				*Messages:*			
η	$::=$	a	nested names	M, N	$::=$	a	name
	$\mid$	$\hat{\hat{}}$	upward		$\mid$	$\mathsf{enter}\langle M, N\rangle$	enter M with N
	$\mid$	$\star$	local		$\mid$	$\mathsf{exit}\langle M, N\rangle$	exit M with N
					$\mid$	$M.N$	path

Prefixes:

$$\pi ::= M \;\big|\; (x_1, \ldots, x_k)^\eta \;\big|\; \langle M_1, \ldots, M_k\rangle^\eta \;\big|\; \overline{\mathsf{enter}}(x, M) \;\big|\; \overline{\mathsf{exit}}(x, M)$$

Processes:

$$\pi ::= \mathbf{0} \;\big|\; P_1 \mid P_2 \;\big|\; (\boldsymbol{\nu} n)P \;\big|\; !\pi.P \;\big|\; M[P] \;\big|\; \pi.P$$

The operators for inactivity, composition, restriction and replication are inherited from mainstream concurrent calculi, most notably the π-calculus [10]. Specific of ambient calculi are the *ambient* construct, $M[P]$, and the *prefix* via capabilities, $M.P$. Meaningless terms such as $\mathsf{enter}\langle n, p\rangle[P]$ will be ruled out by the type system in §2.

We use a number of notational conventions. Parallel composition has the lowest precedence among the operators. The process $M.N.P$ is read as $M.(N.P)$. We write $\langle\tilde{M}\rangle^\eta, (\tilde{x}), (\boldsymbol{\nu}\tilde{n})$ for $\langle M_1, \ldots, M_k\rangle^\eta, (x_1, \ldots, x_k), (\boldsymbol{\nu}\ n_1, \ldots, n_k)$. We omit trailing dead processes, writing M for $M.\mathbf{0}$, $\langle\tilde{M}\rangle$ for $\langle\tilde{M}\rangle.\mathbf{0}$, and $n[\,]$ for $n[\mathbf{0}]$. Finally, the operators $(\boldsymbol{\nu}n)P$, $\overline{\mathsf{enter}}(x, M).P$, $\overline{\mathsf{exit}}(x, M).P$, and $(\tilde{x})^\eta.P$ act as binders for names n, x, and $\tilde{x}$, respectively. The set of *free names* of P, $\mathrm{fn}(P)$, is defined accordingly.

1.1 Reduction and Behavioural Semantics

The dynamics of the calculus is given in the form of a reduction relation $\longrightarrow$ defined as the least relation on processes closed under parallel composition, restriction, and ambient operator which satisfies the following rules.

mobility

(ENTER) $n[\mathsf{enter}\langle m, k\rangle.P_1 \mid P_2] \;\big|\; m[\overline{\mathsf{enter}}(x, k).Q_1 \mid Q_2]$
$\longrightarrow m[n[P_1 \mid P_2] \mid Q_1\{x := n\} \mid Q_2]$

(EXIT) $n[m[\mathsf{exit}\langle n, k\rangle.P_1 \mid P_2] \mid Q] \;\big|\; \overline{\mathsf{exit}}(x, k).R$
$\longrightarrow m[P_1 \mid P_2] \mid n[Q] \mid R\{x := m\}$

communication

(LOCAL) $(\tilde{x}).P \;\big|\; \langle\tilde{M}\rangle.Q \longrightarrow P\{\tilde{x} := \tilde{M}\} \mid Q$

(INPUT n) $(\tilde{x})^n.P \;\big|\; n[\langle\tilde{M}\rangle^{\hat{}}.Q \mid R] \longrightarrow P\{\tilde{x} := \tilde{M}\} \;\big|\; n[Q \mid R]$

(OUTPUT n) $\langle\tilde{M}\rangle^n P \;\big|\; n[(\tilde{x})^{\hat{}}.Q \mid R] \longrightarrow P \;\big|\; n[Q\{\tilde{x} := \tilde{M}\} \mid R]$

structural congruence

(STR CONG) $P \equiv Q \quad Q \longrightarrow R \quad R \equiv S$ implies $P \longrightarrow S$

As customary in process calculi, the *reduction semantics* is based on an auxiliary relation called *structural congruence*, $\equiv$, which brings the participants of a potential interaction to contiguous positions. Its formal definition is similar to that for BA [2]. The reduction rules are divided in two groups: the *mobility rules* and the *communication rules*.

As in [8], mobility requires the ambients involved to agree on a password k. In addition, the target of the move learns the name of the entering ambient as a result of synchronisation. Similar ideas apply to the (EXIT) rule: in particular, the co-exit capability is placed in the target ambient, as in [8]. Unlike *loc. cit.*, however, the co-action does not mention the name of the travelling ambient.

The communication rules are a simplification of those of BA. As in BA communication is *synchronous*, *polyadic*, and, most importantly, *located* and *across* boundaries. In both rules (INPUT n) and (OUTPUT n) the underlying anonymous channel is to be thought of as allocated at the sub-ambient n. In all communication rules we assume that tuples have the same arity, a condition that later will be enforced by the type system.

As to behavioural equivalences we focus on *reduction barbed congruence* [6], a standard equality based on the notions of reduction relation and observability. In ambients the observation predicate $P \downarrow_n$ is used to denote the possibility of process P of interacting with the environment via the ambient n. In MA [3] this happens whenever

$P \equiv (\boldsymbol{\nu}\tilde{m})(n[P_1]|P_2)$, for $n \notin \{\tilde{m}\}$: since no authorisation is required to cross a boundary, the presence of an ambient n at top level denotes a potential interaction between the process and the environment via n. In the presence of co-capabilities, however, the process $(\boldsymbol{\nu}\tilde{m})(n[P_1]|P_2)$ only represents a potential interaction if P_1 can exercise an appropriate co-capability.

Definition 1 (Barbs). *We write* $P\downarrow_n$ *iff* $P \equiv (\nu\boldsymbol{m})(n[\overline{\mathsf{enter}}(x,k).Q \mid R] \mid S)$ *for* $\{n,k\} \cap \{\boldsymbol{m}\} = \emptyset$. *We write* $P\Downarrow_n$ *if* $P \Longrightarrow P'$ *and* $P'\downarrow_n$.

Definition 2. *A relation* $\mathscr{R}$ *is:* ***(i)*** reduction closed *if* $P\mathscr{R}Q$ *and* $P \to P'$ *implies the existence of some* Q' *such that* $Q \Rightarrow Q'$ *and* $P'\mathscr{R}Q'$; ***(ii)*** barb preserving *if* $P\mathscr{R}Q$ *and* $P\downarrow_n$ *implies* $Q\Downarrow_n$.

Definition 3 (Reduction Barbed Congruence). *Reduction barbed congruence, written* $\cong$*, is the largest congruence relation over processes which is reduction closed and barb preserving.*

Notice that in our setting there is at least another significant choice of barb, reflecting the other form of interaction an ambient may engage with the context, viz. via communication. In particular, we could reasonably define that P has a barb at n if and only if $P \equiv (\nu\boldsymbol{m})(n[\langle\cdot\rangle^{\hat{}}.P' \mid Q'] \mid Q'')$ for $n \notin \{\boldsymbol{m}\}$. However, we can show that either choices of barb result in the same equivalence. Let $\downarrow_n^{\mathsf{e}}$ denote the alternative predicate we just define, and $\cong_{\mathsf{e}}$ the resulting equivalence.

Proposition 1. $P \cong Q$ *if and only if* $P \cong_{\mathsf{e}} Q$.

1.2 Algebraic Laws

In the remainder of the section we give a collections of laws which hold in NBA. All these laws can be easily proved using our notion of bisimilarity by exhibiting the appropriate bisimulation relation. In most cases the bisimulation relation contains only the pair of the processes of the law plus the identity relation. The first set of laws is about garbage collection: for $I = J = H = \emptyset$ one also derives $l[\,] = \mathbf{0}$.

Theorem 1 (Algebraic Laws).

Garbage Collection Laws

1. $l[\, \Pi_{i\in I}(\tilde{x}_i)^{n_i}.P_i \mid \Pi_{j\in J}(\tilde{x}_j).P_j \mid \Pi_{h\in H}\langle\tilde{M}\rangle^{m_h}.P_h \,] \cong \mathbf{0}$
2. $l[\, \Pi_{i\in I}(\tilde{x}_i)^{n_i}.P_i \mid \Pi_{j\in J}\langle\tilde{M}_j\rangle.P_j \mid \Pi_{h\in H}\langle\tilde{M}_h\rangle^{m_h}.P_h \,] \cong \mathbf{0}$

Communication Laws *If* $|\tilde{x}|=|\tilde{M}|$ *then:*

1. $l[\, \Pi_{j\in J}\langle\tilde{M}_j\rangle^{\hat{}}\,] \cong \Pi_{j\in J}\, l[\langle\tilde{M}_j\rangle^{\hat{}}\,]$
2. $l[(\tilde{x}).P \mid \langle\tilde{M}\rangle.Q] \cong l[P\{\tilde{x} := \tilde{M}\} \mid Q]$
3. $(\boldsymbol{\nu}l)(\,(\tilde{x})^l.P \mid l[\langle\tilde{M}\rangle^{\hat{}}.Q]\,) \cong (\boldsymbol{\nu}l)(\,P\{\tilde{x} := \tilde{M}\} \mid l[Q]\,)$
4. $m[(\tilde{x})^l.P \mid l[\langle\tilde{M}\rangle^{\hat{}}.Q]] \cong m[P\{\tilde{x} := \tilde{M}\} \mid l[Q]]$

5. *Dual laws of 3 and 4 resulting from exchanging input with output prefixes*

Mobility Laws

1. $(\boldsymbol{\nu} p)(m[\mathsf{enter}\langle n, p\rangle.P] \mid n[\overline{\mathsf{enter}}(x, p).Q]) \;\cong\; (\boldsymbol{\nu} p)(n[Q\{x := m\} \mid m[P]])$
2. $l[m[\mathsf{enter}\langle n, p\rangle.P] \mid n[\overline{\mathsf{enter}}(x, p).Q]] \;\cong\; l[n[Q\{x := m\} \mid m[P]]]$
3. $(\boldsymbol{\nu} p)(n[m[\mathsf{exit}\langle n, p\rangle.P]] \mid \overline{\mathsf{exit}}(x, p).Q) \;\cong\; (\boldsymbol{\nu} p)(m[P] \mid Q\{x := m\})$
4. $l[n[m[\mathsf{exit}\langle n, p\rangle.P]] \mid \overline{\mathsf{exit}}(x, p).Q] \;\cong\; l[m[P] \mid Q\{x := m\}].$ $\square$

The first law says how parallel composition of upward *asynchronous* messages distributes on the ambient construct. The same law holds for BA; in neither case (BA, or NBA) it extends to the case of (upward) output prefixes with non-nil continuation. Law 2 shows that NBA is free of interferences on local communications: it holds in SA but not in MA and BA. The remaining laws are peculiar of NBA and show that, unlike BA, no interference on upward communications is possible.

2 The Type System

The absence of interferences in communication has other interesting payoffs besides those we have discussed in the previous section. Specifically, as we illustrate below, it greatly simplifies the type system over the original definition for BA [2]. The structure of the types is as follows.

Message Types	W	$::=$	$\mathsf{N}[E]$	ambient/password
		$\mid$	$\mathsf{C}[E]$	capability
Exchange Types	E, F	$::=$	shh	no exchange
		$\mid$	$W_1 \times \cdots \times W_k$	tuples ($k \geq 0$)
Process Types	T	$::=$	$[E, F]$	composite exchange

The main difference with respect to the type systems of [2] and [9] is in the structure of ambient types, that are defined here as one-place constructors of the form $\mathsf{N}[E]$, tracing the upward exchanges of ambients with this type. This simplification over previous systems (in which ambient types are two-place constructors) is a direct consequence of the semantics of communication, which guarantees the absence of interferences between the local exchanges of an ambient and the upward exchanges of its nested sub-ambients.

In addition, in the present system types of the form $\mathsf{N}[E]$ also serve as the types of passwords. Hence, $\mathsf{N}[E]$ is indeed the class of *name* types. When used as a password type, $\mathsf{N}[E]$ enables upward exchanges of type E for any process that relies on a password with this type to cross an ambient boundary. There is no type confusion in this double role of name types, as different uses of a name have different imports in the typing rules. An alternative would be to use two different constructors for ambient and password names: this, however, would have the undesired effect of disallowing the use of the same name in the two roles, a feature that is harmless and actually rather convenient in many examples.

Table 1. Good Messages: $\Gamma \vdash M : W$

$$\textsc{(Env }\varnothing\textsc{)}\ \frac{}{\varnothing \vdash \diamond} \qquad \textsc{(Env name)}\ \frac{\Gamma \vdash \diamond \quad a \notin \mathrm{Dom}(\Gamma)}{\Gamma, a : W \vdash \diamond} \qquad \textsc{(Projection)}\ \frac{\Gamma, a : W, \Gamma' \vdash \diamond}{\Gamma, a : W, \Gamma' \vdash a : W} \qquad \textsc{(Path)}\ \frac{\Gamma \vdash M_1 : \mathsf{C}[E_1] \quad \Gamma \vdash M_2 : \mathsf{C}[E_2]}{\Gamma \vdash M_1.M_2 : \mathsf{C}[E_1 \sqcup E_2]}$$

$$\textsc{(Enter)}\ \frac{\Gamma \vdash M : \mathsf{N}[E] \quad \Gamma \vdash N : \mathsf{N}[F] \quad (F \leqslant G)}{\Gamma \vdash \mathsf{enter}\langle M, N\rangle : \mathsf{C}[G]} \qquad \textsc{(Exit)}\ \frac{\Gamma \vdash M : \mathsf{N}[E] \quad \Gamma \vdash N : \mathsf{N}[F] \quad (F \leqslant G)}{\Gamma \vdash \mathsf{exit}\langle M, N\rangle : \mathsf{C}[G]}$$

The remaining types have the same interpretation as in previous type systems. In particular, $\mathsf{C}[E]$ is the type of capabilities exercised within ambients with upward exchange of type E, and $[E, F]$ is the type of processes with local exchange of type E and upward exchanges of type F. The role of the type shh also is similar to that played by the corresponding type in the type system of BA. As in that case, it indicates absence of exchanges, and can be assigned to processes that do not have any local or upward exchange. In addition, in the type systems for NBA, shh provides for a *silent* mode for mobility in a way similar to, but substantially simpler than, the *moded types* of [2]. Specifically, the typing rules guarantee that a boundary crossing, say by ambient n, via a password of type $\mathsf{N}[\mathsf{shh}]$ involves no learning of names. Thus, unless the target ambient knows the name n statically, the use of a $\mathsf{N}[\mathsf{shh}]$ password guarantees that ambients can move irrespective of their upward exchanges.

We proceed with the presentation of the typing rules. Table 1 gives the rules for valid type environments and for messages.

The relation $F \leqslant G$, with F and G exchange types, holds true if and only if $F \in \{\mathsf{shh}, G\}$. The rules (Enter) and (Exit) define the types of capabilities in terms of the type of the component passwords: together with the typing rules for processes, ambients and mobility of Table 3, they construe the types of passwords as *interfaces* for mobility. In particular, if the type F associated with password N is a message type W, then N constrains any ambient relying upon it for mobility to have upward exchanges of type W (cf. rules (Prefix) and (Amb) in Table 3). If instead $F = \mathsf{shh}$, then N enforces no constraint, as the type G in the conclusion of the rule can be any exchange type. Rule (Path) follows the same intuition: it is applicable only when $E_1 \sqcup E_2$ exists, $\sqcup$ being the lub associated with $\leqslant$.

Tables 2 to 4 define the typing of processes. The rules in Table 2 are standard. The rules in Table 3 complement those in Table 1 in governing mobility. Rule (Amb) is standard, and construes the type $\mathsf{N}[E]$ as the interface of ambient M for any process which knows name M: any such process may have sound E exchanges with M, as the process enclosed within M has upward exchanges of this type. The rules for the mobility co-actions provide similar guarantees for the exchanges a process may have with ambients whose name the process learns by exercising the co-capability. In this case, it is the type of the password M that acts as interface: if M has type $\mathsf{N}[\tilde{W}]$, as

Table 2. Good processes I: $\Gamma \vdash P : [E, F]$

$$\text{(PAR)}\ \frac{\Gamma \vdash P : [E,F] \quad \Gamma \vdash Q : [E,F]}{\Gamma \vdash P \mid Q : [E,F]} \qquad \text{(REPL)}\ \frac{\Gamma \vdash P : [E,F]}{\Gamma \vdash\, !P : [E,F]} \qquad \text{(DEAD)}\ \frac{\Gamma \vdash \diamond}{\Gamma \vdash \mathbf{0} : [E,F]} \qquad \text{(NEW)}\ \frac{\Gamma, n : \mathsf{N}[G] \vdash P : [E,F]}{\Gamma \vdash (\boldsymbol{\nu} n{:}\mathsf{N}[G])P : [E,F]}$$

Table 3. Good Processes II (mobility)

$$\text{(AMB)}\ \frac{\Gamma \vdash M : \mathsf{N}[E] \ \ \Gamma \vdash P : [F,E]}{\Gamma \vdash M[P] : [G,H]} \qquad \text{(PREFIX)}\ \frac{\Gamma \vdash M : \mathsf{C}[F] \ \ \Gamma \vdash P : [E,G] \ \ (F \leqslant G)}{\Gamma \vdash M.P : [E,G]}$$

$$\text{(CO-ENTER)}\ \frac{\Gamma \vdash M : \mathsf{N}[\tilde{W}] \ \ \Gamma, x : \mathsf{N}[\tilde{W}] \vdash P : [E,F]}{\Gamma \vdash \overline{\mathsf{enter}}(x, M).P : [E,F]} \qquad \text{(CO-EXIT)}\ \frac{\Gamma \vdash M : \mathsf{N}[\tilde{W}] \ \ \Gamma, x : \mathsf{N}[\tilde{W}] \vdash P : [E,F]}{\Gamma \vdash \overline{\mathsf{exit}}(x, M).P : [E,F]}$$

$$\text{(CO-ENTER-SILENT)}\ \frac{\Gamma \vdash M : \mathsf{N}[\mathsf{shh}] \ \ \Gamma \vdash P : [E,F] \ \ (x \notin \mathrm{fv}(P))}{\Gamma \vdash \overline{\mathsf{enter}}(x, M).P : [E,F]} \qquad \text{(CO-EXIT-SILENT)}\ \frac{\Gamma \vdash M : \mathsf{N}[\mathsf{shh}] \ \ \Gamma \vdash P : [E,F] \ \ (x \notin \mathrm{fv}(P))}{\Gamma \vdash \overline{\mathsf{exit}}(x, M).P : [E,F]}$$

in the rules (CO-ENTER) and (CO-EXIT), we are guaranteed that $\tilde{W}$ is indeed the type of the exchanges of the incoming ambient. If instead the password type is $\mathsf{N}[\mathsf{shh}]$, then no such guarantee can be provided, as it is easily verified by an inspection of the (PREFIX) rule and of the rules for communication in Table 4. To recover soundness, rules (CO-ENTER/EXIT-SILENT) require that no use be made of the continuation process P of the variable x (hence of the name of the incoming ambient, unless the name was known already to P). An alternative sound solution would be to generalise the (CO-ENTER) and (CO-EXIT) rules by (systematically) replacing the type $\tilde{W}$ with a generic exchange type G. Although such an approach would allow to dispense with rules (CO-ENTER/EXIT-SILENT) and (CO-EXIT-SILENT), the resulting system would be less general than the present one, in that ambients using silent passwords (i.e. of type $\mathsf{N}[\mathsf{shh}]$) for mobility would be required to be upward silent. Notice, in fact, that we impose no such constraint: the typing rules only prevent upward exchanges with the processes enclosed in ambients reached by the use of a silent password.

The type system is sound, as expected.

Proposition 2 (Subject Reduction). *If $\Gamma \vdash P : T$ is a derivable judgement in NBA, and $P \longrightarrow Q$, then $\Gamma \vdash Q : T$ is also a derivable judgement.*

Table 4. Good Processes III (input/output)

(INPUT) $\dfrac{\Gamma, \tilde{x}:\tilde{W} \vdash P : [\tilde{W}, E]}{\Gamma \vdash (\tilde{x}:\tilde{W}).P : [\tilde{W}, E]}$	(INPUT $\hat{}$) $\dfrac{\Gamma, \tilde{x}:\tilde{W} \vdash P : [E, \tilde{W}]}{\Gamma \vdash (\tilde{x}:\tilde{W})^{\hat{}}.P : [E, \tilde{W}]}$	(INPUT M) $\dfrac{\Gamma \vdash M : \mathsf{N}[\tilde{W}] \quad \Gamma, \tilde{x}:\tilde{W} \vdash P : [G, H]}{\Gamma \vdash (\tilde{x}:\tilde{W})^{M}.P : [G, H]}$
(OUTPUT) $\dfrac{\Gamma \vdash \tilde{M} : \tilde{W},\ \Gamma \vdash P : [\tilde{W}, E]}{\Gamma \vdash \langle \tilde{M} \rangle.P : [\tilde{W}, E]}$	(OUTPUT $\hat{}$) $\dfrac{\Gamma \vdash \tilde{M} : \tilde{W},\ \Gamma \vdash P : [E, \tilde{W}]}{\Gamma \vdash \langle \tilde{M} \rangle^{\hat{}}.P : [E, \tilde{W}]}$	(OUTPUT N) $\dfrac{\Gamma \vdash N : \mathsf{N}[\tilde{W}],\ \Gamma \vdash \tilde{M} : \tilde{W},\ \Gamma \vdash P : [G, H]}{\Gamma \vdash \langle \tilde{M} \rangle^{N}.P : [G, H]}$

3 Examples

We illustrate the expressive power of NBA and its algebraic theory with several examples. The first is a standard expressiveness test: the encoding of the pi-calculus [10].

3.1 Encoding of Channels

Among the possible solutions, we present the most compact one, which further illustrates the power of our co-actions. We only show the encoding of channels, the remaining clauses are defined homomorphically.

$$\begin{aligned}
\{\!\|\, \bar{c}\langle \tilde{M} \rangle P \,\|\!\} &\triangleq (\boldsymbol{\nu} a)\,(\ c[\langle \tilde{M} \rangle^{\hat{}}.c[\mathsf{exit}\langle c, a \rangle]] \mid \overline{\mathsf{exit}}(x, a).\{\!\|\, P \,\|\!\}\ ) \\
\{\!\|\, c(\tilde{x}) P \,\|\!\} &\triangleq (\tilde{x})^{c}.\{\!\|\, P \,\|\!\}
\end{aligned}$$

Given the direct nature of the encoding, its operational correctness is simple to prove. Let $\gtrsim$ denote the *expansion* relation [1], an asymmetric variant of $\approx$ such that $P \gtrsim Q$ holds if $P \approx Q$, and P has at least as many τ-moves as Q.

Lemma 1 (Operational Correspondence). *If* $P \xrightarrow{\tau} P'$ *then* $\{\!\|\, P \,\|\!\} \xrightarrow{\tau} \gtrsim \{\!\|\, P' \,\|\!\}$. *Conversely, if* $\{\!\|\, P \,\|\!\} \xrightarrow{\tau} Q$*, then there exists* P' *such that* $P \xrightarrow{\tau} P'$ *and* $Q \gtrsim \{\!\|\, P' \,\|\!\}$.

Proof. By transition induction. Expansion relations are derived by law in Theorem 1.3.

The result extends to weak reductions. Based on that, and on the compositionality of $\{\!\|\, \cdot \,\|\!\}$, we can show that the encoding is sound in the following sense.

Theorem 2 (Equational Soundness). *If* $\{\!\|\, P \,\|\!\} \cong \{\!\|\, Q \,\|\!\}$ *then* $P \cong Q$.

3.2 NBA versus BA and SA

The next suite of examples relates NBA and its 'parent' calculi BA and SA, reinforcing our claim that NBA removes non-determinism from BA, and providing more a formal statement about their relative expressive power.

A One-to-One Communication Server. Our first example is a system that represents a server for point-to-point communication.

$$\mathsf{w}(k) \;=\; k[\;\overline{\mathsf{enter}}(x,k).\overline{\mathsf{enter}}(y,k).(!(z)^x.\langle z\rangle^y \mid !(z)^y.\langle z\rangle^x)\;]$$

Here $\mathsf{w}(k)$ is a bidirectional forwarder for any pair of incoming ambients. An agent willing to participate in a point-to-point communication must know the password k and should be implemented as the process $A(a,k,P,Q) = a[\mathsf{enter}\langle k,k\rangle.P \mid \mathsf{exit}\langle k,k\rangle.Q]$, with P performing the expected (upward) exchanges. A complete implementation of the point-to-point server can be then defined as shown below:

$$\mathsf{o2o}(k) = (\boldsymbol{\nu} r)\;(\; r[\langle\,\rangle^{\hat{}}\,] \mid !\,(\,)^r.(\mathsf{w}(k) \mid \overline{\mathsf{exit}}(\_,k).\overline{\mathsf{exit}}(\_,k).r[\langle\,\rangle^{\hat{}}\,])\;)$$

The process $\mathsf{o2o}(k)$ accepts a pair of ambients within the forwarder, provides them with the necessary support of the point-to-point exchange and then lets them out before preparing a new instance of $\mathsf{w}(k)$ for a new protocol session. Given the configuration $\mathsf{o2o}(k) \mid A(k,a_1,P_1,Q_1) \mid \cdots \mid A(k,a_n,P_n,Q_n)$, we are guaranteed that at most one pair of agents can be active within k at any given time (k is locked until the two ambients are inside k). In particular, one has:

$$\begin{aligned}&(\boldsymbol{\nu} k)(\;\mathsf{o2o}(k) \mid A(k,a_1,\langle M\rangle^{\hat{}}.P_1,Q_1) \mid A(k,a_2,(x)^{\hat{}}.P_2\{x\},Q_2) \mid \Pi_{i\in I}A(K,a_i,R_i,S_i)\;)\\ &\qquad\Longrightarrow\cong\;(\boldsymbol{\nu} k)(\;\mathsf{o2o}(k) \mid a_1[P_1 \mid Q_1] \mid a_2[P_2\{x := M\} \mid Q_2] \mid \Pi_{i\in I}A(K,a_i,R_i,S_i)\;)\end{aligned}$$

This says that once (and if) the two agents have reached the forwarder, no other agent knowing the key k can interfere and prevent them from completing their exchange. The equivalence above follows by the 'garbage collection' and the 'mobility' laws of Theorem 1. In particular, once the two ambients are back at top level, the currently active instance of the forwarder k has the form $k[!(z)^{a_1}.\langle z\rangle^{a_2} | !(z)^{a_2}.\langle z\rangle^{a_1}] \cong \mathbf{0}$.

The use of the forwarder to implement a point-to-point communication protocol may at first appear artificial, for it would seem that two ambients wishing to communicate are likely to know their partner's name, and could then interact without intervention of a third party. Indeed, in NBA the example can be simplified with these assumptions. In BA, instead, the knowledge of names still poses a challenge due to possible communication interferences. To illustrate the point, let us consider the following BA coding of the protocol.

$$a[\mathsf{in}\;b.\mathsf{in}\;k.P] \mid b[k[!(x).\langle x\rangle^a \mid !(x)^a.\langle x\rangle] \mid Q]$$

Here Q can read from and write to k to exchange values with P inside a, but it is not obvious what P should do. Clearly, with k as above, it must use local communication to talk with k, and hence with Q. However, this would not avoid interference with its own local exchanges. Not is it clear how to ban such unwanted effect, as there seem to be similar problems with several different implementations of k. For instance, using $k[!(x).\langle x\rangle]$ a and b may end up re-reading their own messages; while with $k[!(x).\langle x\rangle \mid (x)^{\uparrow}.\langle x\rangle^a]$ the upward read by k may mistakenly synchronise with local output in b.

A Print Server. Our next example implements a print server to print jobs arriving off the network in the order of arrival. We give the implementation in steps. First consider the following process that assigns a progressive number to each incoming job.

$$\mathsf{enqueue}(k) = \ (\boldsymbol{\nu} c)\ (\ c[\langle 1\rangle^{\hat{}}\,]\mid !(n)^c.\overline{\mathsf{enter}}(x,k).\langle n\rangle^x.c[\langle n+1\rangle^{\hat{}}\,])$$

The (private) ambient c holds the current value of the counter. The process accepts a job and delivers it the current number. Then, it updates the counter and prepares for the next round. Now, we can show how this can be turned into a print server ($\mathsf{enqueue}(k)$ is defined as above):

$$\begin{array}{ll}
\mathsf{prtsrv}(k) & = k[\ \mathsf{enqueue}(k)\mid \mathsf{print}\ ]\\
\mathsf{print} & = \ (\boldsymbol{\nu} c)\ (\ c[\langle 1\rangle^{\hat{}}\,]\mid !(n)^c.\overline{\mathsf{exit}}(x,n).(data)^x.(P\{data\}\mid c[\langle n+1\rangle^{\hat{}}\,])\\
\mathsf{job}(M,s) & = (\boldsymbol{\nu} p)p[\ \mathsf{enter}\langle s,s\rangle.(n)^{\hat{}}.(\boldsymbol{\nu} q)q[\mathsf{exit}\langle p,n\rangle.\langle M\rangle^{\hat{}}\,]\ ]
\end{array}$$

Process $\mathsf{job}(M,s)$ enters the server $\mathsf{prtsrv}(s)$, it is assigned a number to be used as a password for carrying the job M to the printer process P (note that the use of passwords is critical here).

This situation appears hard to implement with SA(P) or BA. In SA(P) because one would need to know the names of the incoming jobs to be able to assign them their numbers. In BA because dequeuing the jobs (according to the intended FIFO policy) requires a test of the number a job has been assigned, and an atomic implementation of such test appears problematic.

BA $\lesssim$ NBA + Guarded Choice. In order to relate BA and NBA more formally, and support our claim that the only loss of expressiveness in the passage is very directly related the removal of grave communication interferences, we present an encoding of BA into an extended version of NBA. Precisely, we enrich NBA with a limited, focused form of nondeterminism, viz. a sum operator, that we use in the encoding to circumscribed the nondeterminism in communication typical of BA. Besides showing the relationship between the two calculi, this has the further advantage of localising their differences in a single construct.

The encoding is defined parametrically over four names n, mv, pr, pw: n is the name of the ambient (if any) that encloses the process that we are encoding, while the remaining three names are used as passwords. To ease the notation, we use the following shorthands: $\overline{\mathsf{cross}} = !\overline{\mathsf{enter}}(x,\mathsf{mv})\mid !\overline{\mathsf{exit}}(x,\mathsf{mv})$, in $n\ =\ \mathsf{enter}\langle n,\mathsf{mv}\rangle$, and out $n\ =\ \mathsf{exit}\langle n,\mathsf{mv}\rangle$. The encoding is defined in terms of two mutually recursive translations, $\{\!|\cdot|\!\}_n$ and $\langle\!\langle\cdot\rangle\!\rangle_n$, with $\{\!|\cdot|\!\}_n$ leading. The interesting cases are given below, where we assume the implicit side condition that $p,y\notin \mathrm{fn}(P)$.

$$\begin{array}{ll}
\{\!|\,P\,|\!\}_n & = \overline{\mathsf{cross}}\mid\ \langle\!\langle\,P\,\rangle\!\rangle_n\\
\langle\!\langle\,m[P]\,\rangle\!\rangle_n & = m[\{\!|\,P\,|\!\}_m]\\
\langle\!\langle\,(x)^aP\,\rangle\!\rangle_n & = (x)^a\,\langle\!\langle\,P\,\rangle\!\rangle_n\\
\langle\!\langle\,(x)P\,\rangle\!\rangle_n & = (x)\,\langle\!\langle\,P\,\rangle\!\rangle_n + (x)^{\hat{}}\,\langle\!\langle\,P\,\rangle\!\rangle_n +\ \overline{\mathsf{exit}}(y,\mathsf{pw})(x)^y\,\langle\!\langle\,P\,\rangle\!\rangle_n\\
\langle\!\langle\,(x)^{\uparrow}P\,\rangle\!\rangle_n & = (\boldsymbol{\nu} p)p[\mathsf{exit}\langle n,\mathsf{pr}\rangle.(x)^{\hat{}}.\mathsf{enter}\langle n,p\rangle.\langle x\rangle^{\hat{}}\,]\mid \overline{\mathsf{enter}}(y,p)(x)^y\,\langle\!\langle\,P\,\rangle\!\rangle_n\\
\langle\!\langle\,\langle M\rangle^aP\,\rangle\!\rangle_n & = \langle M\rangle^a\,\langle\!\langle\,P\,\rangle\!\rangle_n\\
\langle\!\langle\,\langle M\rangle P\,\rangle\!\rangle_n & = \langle M\rangle\,\langle\!\langle\,P\,\rangle\!\rangle_n + \langle M\rangle^{\hat{}}\,\langle\!\langle\,P\,\rangle\!\rangle_n +\ \overline{\mathsf{exit}}(y,\mathsf{pr})\langle M\rangle^y\,\langle\!\langle\,P\,\rangle\!\rangle_n\\
\langle\!\langle\,\langle M\rangle^{\uparrow}P\,\rangle\!\rangle_n & = (\boldsymbol{\nu} p)p[\mathsf{exit}\langle n,\mathsf{pw}\rangle.\langle M\rangle^{\hat{}}.\mathsf{enter}\langle n,p\rangle.\langle\rangle^{\hat{}}\,]\mid \overline{\mathsf{enter}}(y,p)()^y\,\langle\!\langle\,P\,\rangle\!\rangle_n
\end{array}$$

The remaining cases are defined homomorphically.

The encoding $\{\!|\cdot|\!\}_n$ provides unboundedly many co-capabilities, at all nesting levels, so that ambient mobility in BA is rendered faithfully. As for the translation of the communication primitives, the intuition is the following. The upward exchanges of a BA term are dealt with by the taxi ambients that exit the enclosing ambient n to deliver output (or collect input) and then return to n to unlock the continuation P. The use of restricted names as passwords is essential here for the continuation P to be able to identify its helper taxi ambient without risk of confusion. As for the translation of a local input/output, the three branches of the choice reflect the three possible synchronisations: local, from upward, from a nested ambient. Note, in particular, that the right-most branch of these choices may only match upward requests that are encodings of upward requests: this is guaranteed by the use of the passwords pr and pw that regulate the movements of the read/write taxi ambients. The use of two different passwords ensures that they do not interfere with each other and with other BA ambients' moves, as the latter use mv.

Using the algebraic laws in §1.2 we can show that the encoding is operationally correct in the sense of Lemma 1 for *single-threaded* terms. The single-threadedness hypothesis here, although morally identical to SA's, is adapted to NBA to record that engaging in inter-ambient communications is an activity across ambient boundaries that may as well create grave interferences. For instance, $a[\,\langle x\rangle^{\hat{}} \mid \mathsf{exit}\langle n,k\rangle.P\,]$ cannot be considered single-threaded, as illustrated by, say, the context $\overline{\mathsf{exit}}(x,k).R \mid n[\,(x)^a Q \mid -\,]$. The technical definition is not particularly enlightening, and we omit it in this presentation.

Theorem 3. *If P and Q are single-threaded , then $\{\!|\,P\,|\!\}_n \cong \{\!|\,Q\,|\!\}_n$ implies $P \cong Q$.*

The proof follows the same pattern as the one outlined for the encoding of pi-calculus channels. The additional hypothesis on the source terms P and Q is needed to guarantee the atomicity of the protocol that implements an upward exchange (once the taxi ambient leaves n, we need to make sure that no process causes n to move).

4 Conclusion and Future Work

We presented NBA, a new foundation for the calculus of Boxed Ambients that eliminates grave communication interferences without renouncing to mobility and with no further lost of expressiveness. NBA greatly enhances the distributed nature of BA in the precise sense of removing some obvious challenges to implementation in a distributed scenario. The basic mechanism of NBA is to promote co-capabilities to the role of binding constructs that inform ambients of the incoming ambient's name. Together with a system of password control which verifies the visitor's credentials, this yields an interesting way to learn names dynamically that makes up for most of the expressiveness lost by removing from BA the interference-prone forms of across-boundary communication. Furthermore, the access protocol of NBA has a practical, realistic flavour, at all analogous to the negotiation needed for new computational agents to join existing agent communities, such as networks. We are currently investigating a variant of our calculus where co-capabilities are not binding but they rather perform a renaming of the incoming ambient, as in the Seal calculus [15], giving the recipient full control of the incoming ambients. This variant would be a perfect candidate to compare ambient calculi and Seal. Actually, passwords in NBA plays the same role as channels in Seal: they allow synchronisation for code mobility.

From the theoretical viewpoint, NBA enjoys a rich algebraic theory. Its barbed congruence admits a sound coinductive characterisation built on a LTS semantics. We strongly conjecture that such characterisation is also complete. We presented a type system for NBA whose simplicity and generality relies also on passwords. The mobility types of [9] can be applied in NBA at no additional cost.

Finally, we have characterised the loss of expressiveness by means of several small examples and an encoding of BA to a version of NBA enriched with guarded choice. We plan in future work to investigate encodings without sum, following the lines of, e.g., [11]. We expect only very weak forms of encoding to hold, so as to confirm the present characterisation of the expressiveness gap between BA and NBA.

References

1. S. Arun-Kumar and M. Hennessy. An Efficiency Preorder for Processes. *Acta Informatica*, 29:737–760, 1992.
2. M. Bugliesi, G. Castagna, and S. Crafa. Boxed Ambients. In *Proceedings of TACS'01*, number 2215 in Lecture Notes in Computer Science, 38–63, Springer, 2001.
3. L. Cardelli and A. Gordon. Mobile Ambients. In *Proceedings of FoSSaCS'98*, number 1378 in Lecture Notes in Computer Science, 140–155. Springer, 1998.
4. L. Cardelli and A. Gordon. Equational Properties for Mobile Ambients. In *Proceedings FoSSaCS'99*, volume 1578 Lecture Notes in Computer Science, 212–226, Springer, 1999.
5. S. Crafa, M. Bugliesi, and G. Castagna. Information Flow Security for Boxed Ambients. In *F-WAN: Int. Workshop on Foundations of Wide Area Network Computing*, ENTCS 66.3, Elsevier, 2002.
6. K. Honda and N. Yoshida. On Reduction-based Process Semantics. *Theoretical Computer Science*, 152(2):437–486, 1995.
7. F. Levi and D. Sangiorgi. Controlling Interference in Ambients. In *Proceedins of POPL'00*, 352–364, ACM Press, 2000.
8. M. Merro and M. Hennessy. Bisimulation Congruences in Safe Ambients. In *Proceedings of POPL'02*, 71–80. ACM Press, 2002.
9. M. Merro and V. Sassone. Typing and Subtyping Mobility in Boxed Ambients. In *Proceedings of CONCUR'02*, volume 2421 Lecture Notes in Computer Science, 304–320, Springer, 2002.
10. R. Milner, J. Parrow, and D. Walker. A Calculus of Mobile Processes, Parts I and II. *Information and Computation*, 100:1–77, September 1992.
11. B.C. Pierce and U. Nestmann. Decoding Choice Encodings. *Information and Computation*, 163:1–59, 2000.
12. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, Department of Computer Science, University of Edinburgh, 1992.
13. D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.
14. D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". In *Proceedings of CONCUR'92*, volume 630 of Lecture Notes in Computer Science, 32–46. Springer, 1992.
15. J. Vitek and G. Castagna. Seal: A framework for Secure Mobile Computations. In *Internet Programming Languages*, 1999.

The Seal Calculus Revisited: Contextual Equivalence and Bisimilarity[⋆]

Giuseppe Castagna and Francesco Zappa Nardelli

LIENS (CNRS), 45, rue d'Ulm, 75005 Paris, France
{castagna,zappa}@ens.fr

Abstract. We present a new version of the Seal Calculus, a calculus of mobile computation. We study observational congruence and bisimulation theory, and show how they are related.

1 Introduction

The Seal Calculus is a calculus of mobile computations conceived to model secure programming of large scale distributed systems over open networks. It can be roughly described as the π-calculus [9] with hierarchical location mobility and remote accesses to resources. The original presentation [14] was tailored with an implementation in mind, and offered features that were important from the practical view point (e.g. portals), but unessential for its theoretical study.

In this paper we present a 'revised' version of the Seal Calculus that shares with the original version part of its syntax and all the design guiding principles, while gets rid of the redundant aspects. In particular, portals do not belong to the calculus anymore, the reduction semantics no longer resorts to an auxiliary relation, new security oriented rules handle the extrusion of private names, and the definition of the calculus is parametric on the semantics of remote interaction, thus allowing an easier exploration of the design space.

We concentrate on behavioural equivalences to establish more than a foundation to it, by proving that a bisimulation-based equivalence is sound with respect to weak barbed congruence. Even if the technique used is standard, this work is important because it is the first one that keeps into account agent duplication[1]: avoiding agent duplication hugely simplifies the study of the calculus, but wipes out many properties that characterise real systems.

2 Syntax and Semantics of Seal

The syntax of Seal is reported in Figure 1, where letters u, v, x, y, z range over variables, P, Q, R, S range over processes, and $n \geq 0$. The syntax of processes is the same of Ambient Calculus [2]: the only remark is that replication is guarded. On the other hand, contrary to what happens in Ambients, all interactions take

⋆ Research supported by 'MyThS: Models and Types for Security in Mobile Distributed Systems', EU FET-GC IST-2001-32617.

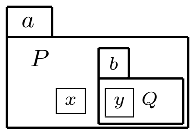
[1] Sangiorgi's research on equivalences for higher order π-calculus [10,11] allows the duplication of processes, but does not account either for agents or for mobility.

Processes			**Actions**			**Locations**		
$P ::=$	$\mathbf{0}$	inactivity	$\alpha ::=$	$\overline{x}^\eta(y_1,\cdots,y_n)$	output	$\eta ::=$	$*$	local
$\mid$	$P \mid P$	composition	$\mid$	$x^\eta(y_1,\cdots,y_n)$	input	$\mid$	$\uparrow$	up
$\mid$	$!\,\gamma.P$	replication	$\mid$	$\overline{x}^\eta\{y\}$	send	$\mid$	z	down
$\mid$	$(\boldsymbol{\nu}\, x)\, P$	restriction	$\mid$	$x^\eta\{y_1,\cdots,y_n\}$	receive	**Guards**		
$\mid$	$\alpha.P$	action				$\gamma ::=$	$\overline{x}^*()$	
$\mid$	$x[P]$	seal				$\mid$	$x^*()$	

Fig. 1. Syntax of the Seal Calculus

place on named localised channels.

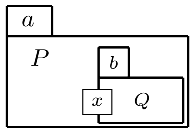
In this work we present two different dialects of Seal. In the first one, called *Located Seal*, channels are *located* inside seals.



Located channels

Channel denotations specify in which seal a channel is located: a channel named x is denoted by x^η, where η is $*$ when the channel is local, is $\uparrow$ when the channel is in the parent, and is n when the channel is in a child seal n. The figure on the left represents a located channels situation with two channels x and y, the former located in a and the latter in b. A synchronisation between P and Q can happen on any of these channels. In order to synchronise on x, process P will use x^* as for P channel x is local, while Q will use $x^\uparrow$ as it is a channel located in the parent. Similarly, to synchronise on y, P will use y^b and Q will use y^*.

In the second dialect, called *Shared Seal*, channels are *shared* between the two communicating agents in parent-child relation, so that the η represents the partner the channel is shared with. Thus, $x^\uparrow$ denotes the channel x shared with the parent seal, x^n the denotes the channel x shared with the child n, while x^* still denotes a local channel. The figure on the left represents a shared channel case. To synchronise, P and Q must use a channel shared between a and b, such as x. For such a synchronisation, P will use x^b as it is a channel shared with b, and Q will use $x^\uparrow$ as it is a channel shared with the parent.



Shared channels

In order to give reduction rules that are parametric on the interaction pattern being used, we introduce two predicates $\mathsf{synch}^S, \mathsf{synch}^L$: **Var** $\times$ **Loc** $\times$ **Loc** $\to$ **Bool**, ranged over by synch. Intuitively, $\mathsf{synch}_y(\eta_1,\eta_2)$ holds if and only if for any channel x an action on x^{η_1} performed in some parent seal may synchronise with a coaction on x^{η_2} performed in a child seal y.

Definition 1. *Let η_1, η_2 be locations and y a variable (a seal name). We define:*
1. $\mathsf{synch}^S_y(\eta_1,\eta_2) \overset{def}{=} (\eta_1 = y \wedge \eta_2 = \uparrow)$ [*Shared* Seal]
2. $\mathsf{synch}^L_y(\eta_1,\eta_2) \overset{def}{=} (\eta_1 = y \wedge \eta_2 = *) \vee (\eta_1 = * \wedge \eta_2 = \uparrow)$ [*Located* Seal]

Channel synchronisation is used for the two possible forms of interaction:

Communication: $\overline{x}^\eta(\boldsymbol{y}).P$ denotes a process waiting to output $\boldsymbol{y}$ on channel x^η and then behave like P; $x^\eta(\boldsymbol{y}).P$ denotes a process waiting to read on channel

x^η some input, say $\boldsymbol{z}$, and then behave like $P\{^{\boldsymbol{z}}/_{\boldsymbol{y}}\}$, that is P in which z_i is substituted for every free occurrence of y_i;

Mobility: $\overline{x}^\eta\{y\}.P$ denotes a process waiting to serialise a child seal named y, send it along channel x^η and then behave like P; $x^\eta\{\boldsymbol{z}\}.P$ denotes a process waiting to receive one seal body along channel x^η, to reactivate n identical copies of it under the names $z_1, \ldots, z_n$ and then behave like P.

The semantics of the Seal Calculus is given in terms of a structural congruence relation and a set of reduction rules. We write $\boldsymbol{x}_n$ or just $\boldsymbol{z}$ to denote the tuple $x_1, \cdots, x_n$, $(\boldsymbol{\nu}\, \boldsymbol{x}_n)$, or just $(\boldsymbol{\nu}\, \boldsymbol{x})$, as an abbreviation for $(\boldsymbol{\nu}\, x_1) \ldots (\boldsymbol{\nu}\, x_n)$, and omit trailing $\mathbf{0}$ processes. We work modulo α-conversion, and require the y_i to be pairwise distinct in the input action. The definition of the set of free variables of a process is standard, except for the receive action that is not a binding operation ($fv(x^\eta\{\boldsymbol{y}\}.P) = fv(P) \cup \boldsymbol{y} \cup \{x\} \cup fv(\eta)$), and coincides with the one in [14].

Definition 2 (Structural Congruence). *The structural congruence relation* $\equiv$ *is the smallest congruence over processes that makes* $(P/\equiv, \mid, \mathbf{0})$ *a commutative monoid and satisfies the following axioms: (1)* $(\boldsymbol{\nu}\, x)\, \mathbf{0} \equiv \mathbf{0}$*; (2)* $(\boldsymbol{\nu}\, x)\, (\boldsymbol{\nu}\, y)\, P \equiv (\boldsymbol{\nu}\, y)\, (\boldsymbol{\nu}\, x)\, P$ *for* $x \neq y$*; (3)* $(\boldsymbol{\nu}\, x)\, (P \mid Q) \equiv P \mid (\boldsymbol{\nu}\, x)\, Q$ *for* $x \notin fv(P)$*; (4)* $!P \equiv P \mid !P$*.*

In the process $(\boldsymbol{\nu}\, \boldsymbol{x})\, P$, we can suppose $x_1, \ldots, x_n$ to be pairwise distinct, and freely permute them (axiom 2 of Definition 2). This implies that the vector $\boldsymbol{x}$ behaves as a set, thus justifying notations such as $(\boldsymbol{\nu}\, \boldsymbol{x} \cap \boldsymbol{y})\, P$ or $(\boldsymbol{\nu}\, \boldsymbol{x} \backslash \boldsymbol{y})\, P$ (where $\cap$ and $\backslash$ denote set-theoretic intersection and difference, with the convention that $(\boldsymbol{\nu}\, \varnothing)\, P = P$).

Definition 2 is the standard π-calculus structural congruence definition. It should be remarked that the Ambient's axiom:

$$(\boldsymbol{\nu}\, x)\, y[\, P\,] \equiv y[\, (\boldsymbol{\nu}\, x)\, P\,] \qquad \text{for } x \neq y \qquad (*)$$

is not (and must not be) used in Seal. This is due to the presence, in Seal, of duplication: it would be semantically unsound to define the processes $(\boldsymbol{\nu}\, x)\, y[\, P\,]$ and $y[\, (\boldsymbol{\nu}\, x)\, P\,]$ as equivalent in the presence of duplication, since if we compose both terms with the copier process $Q = \mathsf{copy}\ y\ \mathsf{as}\ z$ (whose definition can be found in the next page) we obtain: $y[\, (\boldsymbol{\nu}\, x)P\,] \mid Q \rightarrow y[\, (\boldsymbol{\nu}\, x)P\,] \mid z[\, (\boldsymbol{\nu}\, x)P\,]$ and $(\boldsymbol{\nu}\, x)\, y[\, P\,] \mid Q \rightarrow (\boldsymbol{\nu}\, x)\, (y[\, P\,] \mid z[P])$ The first process yields a configuration where seals y and z have each a private channel x, while the second process produces a configuration where y and z share a common channel x.

This observation holds true independently from the Seal framework: the extrusion rule $(*)$ is authorised in Ambient only because its definition does not allow ambients duplication. Among its consequences, it is worth stressing that the extrusion of locally restricted names, when allowed, must be handled explicitly by the reduction rules. The approach we choose is to extrude all locally restricted variables that are communicated to the parent, and no other. This is obtained by the reduction rules shown in Figure 2 to which the usual rules for context and structural congruence reduction must also be added.

$$x^*(\boldsymbol{u}).P \mid \overline{x}^*(\boldsymbol{v}).Q \rightarrow P\{^{\boldsymbol{v}}/_{\boldsymbol{u}}\} \mid Q$$

$$\overline{x}^{\eta_1}(\boldsymbol{v}).P \mid y[(\boldsymbol{\nu}\,\boldsymbol{z})(x^{\eta_2}(\boldsymbol{u}).Q_1 \mid Q_2)] \rightarrow P \mid y[(\boldsymbol{\nu}\,\boldsymbol{z})(Q_1\{^{\boldsymbol{v}}/_{\boldsymbol{u}}\} \mid Q_2)] \qquad \text{if } \boldsymbol{v}\cap\boldsymbol{z}=\varnothing$$

$$x^{\eta_1}(\boldsymbol{u}).P \mid y[(\boldsymbol{\nu}\,\boldsymbol{z})(\overline{x}^{\eta_2}(\boldsymbol{v}).Q_1 \mid Q_2)] \rightarrow (\boldsymbol{\nu}\,\boldsymbol{v}\cap\boldsymbol{z})(P\{^{\boldsymbol{v}}/_{\boldsymbol{u}}\} \mid y[(\boldsymbol{\nu}\,\boldsymbol{z}\setminus\boldsymbol{v})(Q_1 \mid Q_2)])$$

$$x^*\{\boldsymbol{u}\}.P_1 \mid \overline{x}^*\{v\}.P_2 \mid v[Q] \rightarrow P_1 \mid u_1[Q] \mid \cdots \mid u_n[Q] \mid P_2$$

$$\overline{x}^{\eta_1}\{v\}.P \mid v[R] \mid y[(\boldsymbol{\nu}\,\boldsymbol{z})(x^{\eta_2}\{\boldsymbol{u}\}.Q_1 \mid Q_2)] \rightarrow P \mid y[(\boldsymbol{\nu}\,\boldsymbol{z})(Q_1 \mid Q_2 \mid u_1[R] \mid \cdots \mid u_n[R])]$$

$$x^{\eta_1}\{\boldsymbol{u}\}.P \mid y[(\boldsymbol{\nu}\,\boldsymbol{z})(\overline{x}^{\eta_2}\{v\}.Q_1 \mid v[R] \mid Q_2)] \rightarrow P \mid u_1[R] \mid \cdots \mid u_n[R] \mid y[(\boldsymbol{\nu}\,\boldsymbol{z})(Q_1 \mid Q_2)]$$

where $fv(R)\cap\boldsymbol{z}=\varnothing$, $x\notin\boldsymbol{z}$, and $\mathsf{synch}_y(\eta_1,\eta_2)$ holds true.

Fig. 2. Reduction rules

The non-local rules are parametric in synch: different remote interaction patterns are obtained according whether synch is replaced by synch^S (shared channels), or synch^L (located channels).

The first rule describes local communication, which is exactly the same as in the polyadic π-calculus. The second rule describes the communication of a tuple $\boldsymbol{v}$ from a parent to its child y, which takes place provided that η_1 and η_2 and y match a synchronisation pattern, channel x is not locally restricted (i.e., $x\notin\boldsymbol{z}$), and no communicated variable is captured (i.e., $\boldsymbol{v}\cap\boldsymbol{z}=\varnothing$). The third rule is where extrusion of local restrictions of communicated variables takes place, as it corresponds to the case where a child y communicates to its parent a vector $\boldsymbol{v}$ of names. As for all remote synchronisations η_1 and η_2 and y must allow synchronisation and x must not be locally restricted (i.e., $x\notin\boldsymbol{z}$). Local (in y) restrictions of variables that are communicated to the parent (i.e., the variables in $\boldsymbol{v}\cap\boldsymbol{z}$) are extruded while the restrictions of the other variables (i.e., the variables in $\boldsymbol{z}\setminus\boldsymbol{v}$) stay in y.

The fourth rule states that in local mobility the body of the seal specified by the send action is copied as many times as specified by the receive action. This allows an easy implementation of operations like the copy of a seal $(\mathsf{copy}\,x\,\mathsf{as}\,z).P \overset{def}{=} (\boldsymbol{\nu}\,y)(\overline{y}^*\{x\} \mid y^*\{x,z\}.P)$ and its destruction $(\mathsf{destroy}\,x).P \overset{def}{=} (\boldsymbol{\nu}\,y)(\overline{y}^*\{x\} \mid y^*\{\ \}.P)$. The fifth rule states that a seal can be moved inside a child y provided that (*i*) η_1 and η_2 are y-corresponding locations, (*ii*) channel x is not locally restricted (i.e., $x\notin\boldsymbol{z}$), and (*iii*) no variable free in the moved process is captured (i.e., $fv(R)\cap\boldsymbol{z}=\varnothing$).

The last rule breaks the analogy between communication and mobility rules, and differs from semantics given in [14], as no extrusion is performed. In fact, the last rule requires that the body of the moved seal does not contain free any locally restricted variable (i.e., $fv(R)\cap\boldsymbol{z}=\varnothing$). This implies that all variables free in an exiting seal must already be known by the parent, either because they are non-local or because they were previously communicated to it. There are two reasons for choosing this more restrictive solution. First, this approach requires that private names are explicitly exported, giving the programmer a tighter control on local resources. Second, in a perspective implementation locally restricted channels would correspond to local variables. Thus, in case of mobility the free variables are handles that can be accessed only if some explicit reference is passed along with them. What we require here to be explicit, would be in any case implicit in the implementation.

3 Equivalences

In this section we study a semantic equivalence theory for the Seal Calculus. The goal is to determine what an 'adequate' semantic equivalence relation for agents should be. For example in [2,3] Cardelli and Gordon introduce and study a Morris-style contextual equivalence for Mobile Ambients according to which the process $(\boldsymbol{\nu}\, n)\; n[\, P\,]$ cannot be distinguished from the inactive process $\mathbf{0}$ when n does not occur free in P. The intuition is that since the name n is unknown both inside and outside the process, no other ambient can exert a capability on it. Thus it is as if the ambient n did not exist. This is summarised by the so-called *perfect firewall equation* which states that if $n \notin \mathit{fv}(P)$, then $(\boldsymbol{\nu}\, n)\; n[\, P\,] \simeq \mathbf{0}$. One may wonder whether this firewall is so perfect. Indeed the equation above does not ensure that n will not have any interaction with the surrounding context. As a matter of fact, n can enter another ambient that runs in parallel or exit the ambient it resides in. In other words n has total mobility freedom. More formally this means that if for example we consider the *commitment semantics* defined for Mobile Ambients in [4], then the process $(\boldsymbol{\nu}\, n)\; n[\, P\,]$ may emit actions such as $\mathtt{in}\, m$ and $\mathtt{out}\, m$ [2]. This means that no reasonable bisimilarity relation that observes mobility capabilities will equate $\mathbf{0}$, that does not emit anything, with $(\boldsymbol{\nu}\, n)\; n[\, P\,]$. It is thus legitimate to wonder about the adequacy of the observation used to define $\simeq$.

A first answer to the question of what an appropriate notion of equivalence should be has been recently proposed for Ambients by Merro and Hennessy in a work [8] that strives towards our same goals and from which this section is deeply inspired. Merro and Hennessy work starts from Sangiorgi's observation in [12] that the algebraic theory of Ambients is poor. The goal of [8] is thus to modify Mobile Ambients so to endow them with an equational theory that is (*i*) richer, (*ii*) reasonable, (*iii*) adequate, and (*iv*) practicable. What do these four properties mean? Richer: that it proves equivalences more interesting than the simple structural congruence relation; reasonable: that it is a contextual equivalence that preserves reductions and some simple observational property; adequate: that it is invariant to different choices of observations (technically, of *barbs*); practicable: that it can be expressed in terms of bisimulation, whose coinductive nature ensures the existence of powerful proof techniques.

A first step in this direction was done by Levi and Sangiorgi [7] who extended Ambients by coactions. A reduction takes place only if an action synchronises with a corresponding coaction, which yields to a more satisfactory equational theory. Nevertheless we are once more in the presence of a contextual equivalence which does not enjoy the last two properties. In [8] Merro and Hennessy extend

[2] More precisely, according to the system in [4] a process of the form $(\boldsymbol{\nu}\, n)\; n[\, P\,]$ may emit $\overline{\mathit{enter}\ m}$ (and thus enter in a sibling ambient m) and $\mathit{exit}\ m$ (and thus exit from a surrounding ambient m).

(and modify) the work of [7] by adding to Ambients, besides coactions, also some *passwords*: an action and the corresponding coaction synchronise only if they possess the same password. Then, Merro and Hennessy define a bisimulation-based equivalence that is invariant for a large choice of observations. In other terms, they show that their extension enjoys the four required properties.

It is quite interesting to notice that all these modifications, proposed in order to endow Mobile Ambients with more sensible equational theories, make it closer and closer to the Seal Calculus: [7] requires mobility to be the consequence of a process synchronisation; [8] simply requires that the mobility takes place on channels (as Merro and Hennessy's passwords can be easily assimilated to channels)[3]. The very last step that distinguishes these Ambient variations from Seal is that Seal uses objective mobility — the agent is sent by the surrounding environment — while in Ambient-based calculi mobility is subjective — the agent sends itself — (as an aside, note that objective moves have also been added to Ambients by Cardelli, Ghelli and Gordon [1] in order to have more refined typings). So it seems quite natural that results similar to those of Merro and Hennessy can be stated for Seals without requiring any modification of its definition. This is what we do in this section, which constitutes the technical core and the difficult part of this work. Thus we start to define in Section 3.1 a labelled transition system and prove its equivalence with the reduction semantics of the previous section. Then in Section 3.2 we define a contextual equivalence (a barbed congruence) and a bisimilarity relation based on the previous labelled transition systems. We prove that the bisimilarity is a congruence and is sound with respect to (i.e. contained in) the contextual equivalence. So we have a notion of equivalence that nearly satisfies the four requirement we stated. To have the same results as in [8] it remains to prove the completeness of the bisimilarity. Unfortunately this seems to require non-trivial modifications as we explain at the end of the presentation.

3.1 Labelled Transition System

If we compare the study of equivalences for the Seal Calculus with the one done by Merro and Hennessy for the Ambient Calculus, then Seal presents two main difficulties. First, and foremost, the use of objective, rather than subjective, moves requires a three-party synchronisation (like in [6]) that introduces further complexity as it requires some intermediate ad hoc transitions. Second, the presence of channelled synchronisations together with the stricter discipline of Seal on private names make the handling of extrusion much more difficult.

For the rest of this section we focus on the *shared* version of Seal. In particular the lts and the Definition 6 is sensible only for shared channels as some modifications are needed to account for the located variant[4].

In Figure 4 we report the labelled transition system (lts) for the *Shared* Seal.

[3] Merro and Hennessy also modify Levi and Sangiorgi's calculus so that the coaction of an `out` must be placed exactly as a receive action in Seal.

[4] In *Located* Seal the two subprocesses in $x^*(y) \mid (\boldsymbol{\nu}\, a)\; a[\,\overline{x}^{\uparrow}(z)\,]$ can synchronise causing the extrusion of $(\boldsymbol{\nu}\, a)$, while in *Shared* Seal they cannot.

Labels			**Activities**			**Locations**	
ℓ ::=	τ	internal action	a ::=	$x^{\eta}(\boldsymbol{y})$	input	γ ::=	$*$ here
|	P_z	seal freeze	|	$\overline{x}^{\eta}(\boldsymbol{y})$	output	|	z inside z
|	P^z	seal chained	|	$\overline{x}^{\eta}\{\!|y|\!\}$	send		
|	$\gamma[a]$	activity a at γ	|	$\overline{x}^{\eta}\{\!|P|\!\}$	capsule		
			|	$x^{\eta}\{\!|P|\!\}$	receive		
			|	x^{η}_{z}	lock		

The free names of a label, $fv(\ell)$, are defined according to the following rules:

$$fv(\tau) = \varnothing \qquad fv(P_z) = fv(P^z) = \{z\} \cup fv(P) \qquad fv(\gamma[a]) = fv(\gamma) \cup fv(a)$$
$$fv(x^{\eta}(\boldsymbol{y})) = fv(\overline{x}^{\eta}(\boldsymbol{y})) = \{x, \boldsymbol{y}\} \cup fv(\eta) \qquad fv(\overline{x}^{\eta}\{\!|y|\!\}) = \{x, y\} \cup fv(\eta)$$
$$fv(\overline{x}^{\eta}\{\!|P|\!\}) = fv(x^{\eta}\{\!|P|\!\}) = \{x\} \cup fv(\eta) \cup fv(P) \qquad fv(x^{\eta}_{z}) = \{x, z\} \cup fv(\eta)$$

The label τ is the standard silent label indicating internal synchronisation. The label P_z denotes a seal z running P that *freezes* itself, in order to be moved. The label P^z denotes a partial synchronisation: a process willing to move a seal named z and a process willing to receive a seal with body P synchronised, and are now looking for the frozen seal to be moved. An activity $\gamma[a]$ denotes the offer of a visible interaction from a process located at γ.

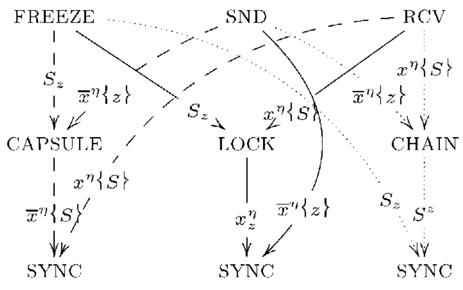


Fig. 3. Synchronisation paths

More in detail, the $x^{\eta}(y)$ label denotes the offer of the input of the value y over channel x tagged by η, the $\overline{x}^{\eta}(y)$ label denotes the offer of the output of the value y over channel x tagged by η, the $\overline{x}^{\eta}\{\!|y|\!\}$ label denotes the offer of sending a seal named y over channel x tagged by η, and the $x^{\eta}\{\!|P|\!\}$ label denotes the offer of receiving the seal body P over channel x tagged by η. The label $\overline{x}^{\eta}\{\!|P|\!\}$ represents the action of *serialising* a seal: its emission indicates that a process willing to send a seal over a channel found it, serialised it, and is now waiting for synchronising with a receiver process. The label x^{η}_{z}, too, denotes a partial synchronisation: a process willing to receive a seal at x^{η} synchronised with the corresponding frozen seal of name z, and is now looking for a sender. If γ is $*$, then the activity takes place at the current level, if γ is a name z then the activity takes place inside a seal named z. Not all activities are visible outside the current seal, and none is visible outside the containing seal. A schema describing the possible synchronisation paths for mobility is reported in Figure 3 (localities and communications have been omitted for clarity). The $\curlyvee$ relation is used to keep track of the couple of labels that match to generate a τ transition.

Definition 3. *Let* $\curlyvee$ *be the smallest binary* symmetric *relation on labels containing the following relation:*

$$\{\ (\ \gamma_1[\overline{x}^{\eta_1}(\boldsymbol{y})]\ ,\ \gamma_2[x^{\eta_2}(\boldsymbol{y})]\ )\mid \mathcal{M}\ \}\ \cup\ \{\ (\ \gamma_1[\overline{x}^{\eta_1}\{\!|S|\!\}]\ ,\ \gamma_2[x^{\eta_2}\{\!|S|\!\}]\ )\mid \mathcal{M}\ \}$$
$$\cup\ \{\ (\ \gamma_1[x^{\eta_1}_{z}]\ ,\ \gamma_2[\overline{x}^{\eta_2}\{\!|z|\!\}]\ )\mid \mathcal{M}\ \}\ \cup\ \{\ (\ S_z\ ,\ S^z\ )\ \}$$

where $\mathcal{M} \stackrel{def}{=} (\gamma_1 = \eta_1 = \gamma_2 = \eta_2 = *) \vee (\gamma_1 = * \wedge \mathsf{synch}_{\gamma_2}(\eta_1, \eta_2)) \vee (\gamma_2 = * \wedge \mathsf{synch}_{\gamma_1}(\eta_2, \eta_1))$.

The labelled transition relation has the form $A \vdash P \xrightarrow{\ell} P'$ where A is a finite set of names and $fv(P) \subseteq A$; it has to be read as 'in a state where names in A may be known by process P and by its environment, the process P can perform ℓ and become P''. This presentation, borrowed from [13], allows us to drop many side conditions dealing with the extrusion of names. The lts is reported in Figure 4. It defines an *early* semantics, as rules (IN) and (RCV) show. This avoids explicitly dealing with process substitutions and is well suited to study bisimulation. The conditions $\gamma = * \Rightarrow \eta \neq \uparrow$ on rule (OPEN CAPSULE) and $fv(S) \subseteq A$ on rule (SEAL LABEL) guarantee that moving a seal body outside the current seal cannot extrude a local name.

The following theorem states the equivalence between the lts and the semantics in chemical style.

Theorem 1. *Let P be a process: (i) if $fv(P) \subseteq A$ and $A \vdash P \xrightarrow{\tau} Q$, then $P \rightarrow Q$, and (ii) if $P \rightarrow Q$ then there exists $A \supseteq fv(P)$ such that $A \vdash P \xrightarrow{\tau} Q'$, where $Q' \equiv Q$.*

3.2 Equivalence Relations

We next define a contextual equivalence for Seal processes. This equivalence is based on the observation of the presence at top level of a seal whose name is unrestricted. Such an observation, due to Cardelli and Gordon [2,3], can be interpreted as the ability of the top-level process to interact with that seal.

We write $\rightarrow^*$ and $\Rightarrow$ for the reflexive and transitive closure of $\rightarrow$ and $\xrightarrow{\tau}$, respectively. We have the following definitions:

Definition 4 (Barbs). *We write $P \downarrow n$ if and only if there exist Q, R, $\boldsymbol{x}$ such that $P \equiv (\boldsymbol{\nu}\, \boldsymbol{x})\, (n[\,Q\,] \mid R)$ where $n \notin \boldsymbol{x}$. We write $P \Downarrow n$ if there exists P' such that $P \rightarrow^* P'$ and $P' \downarrow n$.*

Definition 5 (Barbed Congruence). *Barbed congruence $\cong$ is the largest congruence relation over processes that (i) is reduction closed, that is: if $P \cong Q$ and $P \rightarrow P'$, then there exists Q' such that $Q \rightarrow^* Q'$ and $P' \cong Q'$; (ii) preserves barbs, that is: $P \cong Q$ and $P \downarrow n$ implies $Q \Downarrow n$.*

As a first application of these definitions we can show that $P = (\boldsymbol{\nu}\, x)\, n[\,R\,]$ is not equivalent to $Q = n[\,(\boldsymbol{\nu}\, x)\, R\,]$, and prove in this way the unsoundness of rule $(*)$ given in Section 2. It just suffices to take $R = \overline{y}^{\uparrow}(x) \mid x^{\uparrow}()$ and to consider the context $\mathcal{C}[-] = \mathsf{copy}\, n\, \mathsf{as}\, m.y^{n}(u).\overline{u}^{m}().b[\,] \mid [-]$, where b is fresh. Then $\mathcal{C}[P] \rightarrow^* P'$ and $P' \downarrow b$ while there is no Q' such that $\mathcal{C}[Q] \rightarrow^* Q'$ and $Q' \Downarrow b$.

As the above example shows, contextual equivalence is useful to prove that two processes are *not* equivalent (it suffices to find a context that differentiate them) but it is unfit to prove the equivalence of processes. To that end we seek for a coinductive characterisation of the barbed congruence above.

Congruence

(PAR)
$$\frac{A \vdash P \xrightarrow{\ell} P'}{A \vdash P \mid Q \xrightarrow{\ell} P' \mid Q}$$

(RES) $\forall i,\ x_i \notin fv(\ell)$
$$\frac{A \cdot \boldsymbol{x} \vdash P \xrightarrow{\ell} P'}{A \vdash (\boldsymbol{\nu}\, \boldsymbol{x})\, P \xrightarrow{\ell} (\boldsymbol{\nu}\, \boldsymbol{x})\, P'}$$

(BANG)
$$\frac{}{A \vdash !\gamma.P \xrightarrow{\gamma} P \mid !\gamma.P}$$

(OPEN COM) $y, \eta, \gamma \notin \boldsymbol{u}$
$$\frac{A \cdot \boldsymbol{u} \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}(\boldsymbol{x})]} P'}{A \vdash (\boldsymbol{\nu}\, \boldsymbol{u})\, P \xrightarrow{\gamma[\overline{y}^{\eta}(\boldsymbol{x})]} (\boldsymbol{\nu}\, \boldsymbol{u} \setminus \boldsymbol{x})\, P'}$$

(OPEN FREEZE) $z \notin \boldsymbol{u}$
$$\frac{A \cdot \boldsymbol{u} \vdash P \xrightarrow{S_z} P'}{A \vdash (\boldsymbol{\nu}\, \boldsymbol{u})\, P \xrightarrow{S_z} (\boldsymbol{\nu}\, \boldsymbol{u} \setminus fv(S))\, P'}$$

(SEAL TAU)
$$\frac{A \vdash P \xrightarrow{\tau} P'}{A \vdash x[\,P\,] \xrightarrow{\tau} x[\,P'\,]}$$

(OPEN CAPSULE) $y, \eta, \gamma \notin \boldsymbol{u}$; if $\gamma = *$ then $\eta \neq \uparrow$
$$\frac{A \cdot \boldsymbol{u} \vdash P \xrightarrow{\gamma[\overline{y}^{\eta}\lbag S \rbag]} P'}{A \vdash (\boldsymbol{\nu}\, \boldsymbol{u})\, P \xrightarrow{\gamma[\overline{y}^{\eta}\lbag S \rbag]} (\boldsymbol{\nu}\, \boldsymbol{u} \setminus fv(S))\, P'}$$

(SEAL LABEL) $fv(S) \subseteq A, \exists \eta'.\mathsf{synch}_x(\eta', \eta)$
$$\frac{A \vdash P \xrightarrow{*[a]} P' \quad a \in \{y^{\eta}(\boldsymbol{z}), \overline{y}^{\eta}(\boldsymbol{z}), y^{\eta}\lbag Q \rbag, \overline{y}^{\eta}\lbag S \rbag\}}{A \vdash x[\,P\,] \xrightarrow{x[\,a\,]} x[\,P'\,]}$$

Communication

(OUT)
$$\frac{}{A \vdash \overline{x}^{\eta}(\boldsymbol{y}).P \xrightarrow{*[\overline{x}^{\eta}(\boldsymbol{y})]} P}$$

(IN)
$$\frac{}{A \vdash x^{\eta}(\boldsymbol{y}).P \xrightarrow{*[x^{\eta}(\boldsymbol{v})]} P\{{}^{\boldsymbol{v}}\!/\!{}_{\boldsymbol{y}}\}}$$

Mobility

(SND)
$$\frac{}{A \vdash \overline{x}^{\eta}\lbag y \rbag.P \xrightarrow{*[\overline{x}^{\eta}\lbag y \rbag]} P}$$

(RCV)
$$\frac{}{A \vdash x^{\eta}\lbag \boldsymbol{y} \rbag.P \xrightarrow{*[x^{\eta}\lbag Q \rbag]} P \mid y_1[\,Q\,] \mid \cdots \mid y_n[\,Q\,]}$$

(CAPSULE)
$$\frac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{*[\overline{x}^{\eta}\lbag z \rbag]} Q'}{A \vdash P \mid Q \xrightarrow{*[\overline{x}^{\eta}\lbag S \rbag]} P' \mid Q'}$$

(LOCK) $\gamma = \eta = *$ or $\exists \eta'.\mathsf{synch}_{\gamma}(\eta', \eta)$
$$\frac{A \vdash P \xrightarrow{S_z} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta}\lbag S \rbag]} Q'}{A \vdash P \mid Q \xrightarrow{\gamma[x_z^{\eta}]} (\boldsymbol{\nu}\, fv(S) \setminus A)\, (P' \mid Q')}$$

(FREEZE)
$$\frac{}{A \vdash x[\,P\,] \xrightarrow{P_x} \mathbf{0}}$$

(CHAIN) $\gamma = \eta_1 = \eta_2 = *$ or $\mathsf{synch}_{\gamma}(\eta_1, \eta_2)$
$$\frac{A \vdash P \xrightarrow{*[\overline{x}^{\eta_1}\lbag y \rbag]} P' \quad A \vdash Q \xrightarrow{\gamma[x^{\eta_2}\lbag S \rbag]} Q'}{A \vdash P \mid Q \xrightarrow{S^y} P' \mid Q'}$$

Synchronisation

(SYNC) $\ell_1 \curlyvee \ell_2$
$$\frac{A \vdash P \xrightarrow{\ell_1} P' \qquad A \vdash Q \xrightarrow{\ell_2} Q'}{A \vdash P \mid Q \xrightarrow{\tau} (\boldsymbol{\nu}\, (fv(\ell_1) \cup fv(\ell_2)) \setminus A)\, (P' \mid Q')}$$

The symmetric rules for (PAR), (CAPSULE), (LOCK), and (CHAIN) are omitted.
Notation: $A \cdot \boldsymbol{u}$ is defined as $A \cup \boldsymbol{u}$ if A and $\boldsymbol{u}$ are disjoint, it is undefined otherwise.

Fig. 4. Labelled transition system for shared channels

First of all, remark that the exposure of a barb corresponds to the emission of a (FREEZE) label in the lts:

Lemma 1. *$P \downarrow n$ iff $A \vdash P \xrightarrow{Q_n} P'$ for some P', Q, and A, with $fv(P) \subseteq A$.*

The lemma above shows that the observation used in the contextual equivalence is insensitive to the particular process Q occurring in the label of the labelled transition. Thus, we expect a coinductive characterisation of this equivalence not to be strict in matching (higher-order) labels in which processes occur. As a matter of facts, when agents can be communicated, requiring processes appearing in matching labels to be equal is overly restrictive (as, for instance, in our case $x[\,\mathbf{0}\,]$ and $x[\,y^z()\,]$ would then not be equivalent). On the other hand requiring them to be bisimilar is source of problems when agent mobility requires extrusions of names.

To escape this *impasse* we resort to the intuition underlying the definition of Sangiorgi's *delay bisimilarity* for HOπ ([10], [11]), and require that the outcomes of two bisimilar processes emitting higher order transitions are equivalent with respect to every possible interaction with a context. To that end we introduce the definition of *receiving contexts*. These are processes parametric in two processes X and Y, where Y may get replicated. Receiving contexts represents all the possible outcomes that may result from the migration of a seal, where the parameter processes X and Y stand, respectively, for the residuum of the process that sent the seal and for the body of the moved seal.

Definition 6 (Receiving Context). *Given two processes X and Y where $fv(X) \subseteq A$, a* receiving context *$\mathcal{D}^A_{\gamma,\eta}[X,Y]$ and its* associated environment *$A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ are respectively a process and a context defined as:*

if $\gamma,\eta = *,*$, or $\gamma,\eta = z,\uparrow$ and $fv(Y) \subseteq A$, then for all $\boldsymbol{z}$ such that $fv(\mathcal{D}^A_{\gamma,\eta}[X,Y]) \subseteq A \cup \boldsymbol{z}$

$$(\boldsymbol{\nu}\, fv(Y) \setminus A)(\ X \mid z_1[\,Y\,] \mid \cdots \mid z_n[\,Y\,]\ )$$

and its associated environment $A_{\mathcal{D}^A_{*,*}[X,Y]}$ is $A \cup \boldsymbol{z}$;
if $\gamma,\eta = *,z$, then for all $\boldsymbol{v},\boldsymbol{z}$, and U such that $fv(Y) \cap \boldsymbol{v} = \varnothing$, and $fv(\mathcal{D}^A_{*,z}[X,Y]) \subseteq A \cup (\boldsymbol{z} \setminus \boldsymbol{v})$

$$(\boldsymbol{\nu}\, fv(Y) \setminus A)(\ X \mid z[\,(\boldsymbol{\nu}\,\boldsymbol{v})(\ z_1[\,Y\,] \mid \cdots \mid z_n[\,Y\,] \mid U\ )\,]\ )$$

and its associated environment $A_{\mathcal{D}^A_{*,z}[X,Y]}$ is $A \cup (\boldsymbol{z} \setminus \boldsymbol{v})$;
if $\gamma,\eta = *,\uparrow$, then for all $\boldsymbol{v}, U, z$, such that $fv(Y) \cap \boldsymbol{v} = \varnothing$, $fv(Y) \subseteq A$, and $fv(\mathcal{D}^A_{\gamma,\uparrow}[X,Y]) \subseteq (A \setminus \boldsymbol{v}) \cup \boldsymbol{z}$

$$z[\,(\boldsymbol{\nu}\,\boldsymbol{v})\,(X \mid U)\,] \mid z_1[\,Y\,] \mid \cdots \mid z_n[\,Y\,]$$

and its associated environment $A_{\mathcal{D}^A_{\gamma,\uparrow}[X,Y]}$ is $(A \setminus \boldsymbol{v}) \cup \boldsymbol{z}$[5].

[5] Note that the associated environment is defined in term of the process rather than of the context, as its definition depends on the possibly fresh variables $\boldsymbol{z}$.

We write $\mathcal{D}^A[X,Y]$ when we quantify over all γ,η and abbreviate $A_{\mathcal{D}^A_{\gamma,\eta}[X,Y]}$ by $A_{\mathcal{D}}$ when no ambiguity arises.

Receiving contexts are then used to compare higher-order labelled transitions:

Definition 7 (Hoe Bisimilarity).

Let hoe bisimilarity $\sim$ be the largest family of symmetric relations indexed by finite sets of names such that each $\sim_A$ is a binary relation over $\{P \mid fv(P) \subseteq A\}$ and for all $P \sim_A Q$ the following conditions hold:

1. *if $A \vdash P \xrightarrow{\tau} P'$ then there exists a Q' such that $A \vdash Q \Rightarrow Q'$ and $P' \sim_A Q'$;*
2. *if $A \vdash P \xrightarrow{\ell} P'$ and $\ell \in \{\ \gamma[x^\eta(\boldsymbol{y})], \gamma[\overline{x}^\eta(\boldsymbol{y})], *[\overline{x}^\eta\{\!\!\{ y \}\!\!\}], \gamma[x^\eta\{\!\!\{ S \}\!\!\}], S^z, \gamma[x^\eta_z]\ \}$, then there exists a Q' such that $A \vdash Q \Rightarrow\!\xrightarrow{\ell} Q'$ and $P' \sim_{A\cup fv(\ell)} Q'$;*
3. *if $A \vdash P \xrightarrow{R_z} P'$ then there exist Q', S such that $A \vdash Q \Rightarrow\!\xrightarrow{S_z} Q'$ and for all admissible contexts $\mathcal{D}^A[-,-]$ it holds $\mathcal{D}^A[P',R] \ \sim_{A_\mathcal{D}}\ \mathcal{D}^A[Q',S]$;*
4. *if $A \vdash P \xrightarrow{\gamma[\overline{x}^\eta\{\!\!\{ R \}\!\!\}]} P'$ then there exist Q', S such that $A \vdash Q \Rightarrow\!\xrightarrow{\gamma[\overline{x}^\eta\{\!\!\{ S \}\!\!\}]} Q'$ and for all admissible contexts $\mathcal{D}^A_{\gamma,\eta}[-,-]$ it holds $\mathcal{D}^A_{\gamma,\eta}[P',R] \ \sim_{A_\mathcal{D}}\ \mathcal{D}^A_{\gamma,\eta}[Q',S]$;*
5. *for all substitutions σ, $P\sigma \sim_{A\sigma} Q\sigma$;*

where a context $\mathcal{D}^A[-,-]$ is admissible *if both process substitutions $\mathcal{D}^A[P',R]$ and $\mathcal{D}^A[Q',S]$ are well-formed (i.e. no name capture arises).*

The first two cases of Definition 7 handle all low-order labels, as well as labels originating from receive actions: these do not deserve a special treatment because our early semantics implicitly tests all possible interactions. The cases 3. and 4. check mobility, by testing against all possible outcomes after a mobility interaction with a context.

The most important result of this work is the soundness of bisimilarity:

Theorem 2 (Soundness). *Hoe bisimilarity is sound with respect to barbed congruence: if $P \sim_A Q$ for some A, then $P \cong Q$.*

The proof of this theorem is a consequence of the following lemma.

Lemma 2. *Hoe bisimilarity is a congruence.*

As an application of this theory let us go back to the perfect firewall equation at the beginning of this section. It is quite easy to prove that $(\boldsymbol{\nu}\, x)\, x[\, P\,] \sim \mathbf{0}$ as it suffices to exhibit the following bisimulation $\mathcal{B} = \bigcup_A \mathcal{B}_A$, where $\mathcal{B}_A = \{((\boldsymbol{\nu}\, x)\, x[\, Q\,], \mathbf{0}) \mid P \boldsymbol{\rightarrow}^* Q\} \cup \{(\mathbf{0}, (\boldsymbol{\nu}\, x)\, x[\, Q\,]) \mid P \boldsymbol{\rightarrow}^* Q\}$ if $fv(P) \subseteq A$, and empty otherwise. The use of hoe bisimilarity ensures us that this firewall *is* perfect, as it cannot emit anything but the silent label. The soundness of bisimilarity implies $(\boldsymbol{\nu}\, x)\, x[\, P\,] \cong \mathbf{0}$.

Open issues. The definition of hoe bisimilarity for located channels does not seem worth to be pursued. It is easy to see that with localised channels hoe bisimilarity is not a congruence (the problem being the extrusion of seal names corresponding to $x[a]$ labels), and while it is not difficult to make the needed modifications, this does not seem interesting since the resulting equivalence would be too strong (e.g., $(\boldsymbol{\nu}\, x)\, x[\,P\,] \not\sim \mathbf{0}$).

A more interesting problem is that Hoe bisimilarity is not complete with respect to barbed congruence. Although is is easy to construct a counterexample by exploiting the fact that weak transitions $\Rightarrow\xrightarrow{\ell}$ do not allow τ moves a visible action, the problem is deeper than that. First, since Seal Calculus is an extension of the π-calculus and in the π-calculus the matching operator is necessary to completeness, then this same operator may be required also in Seal; second, in the three party synchronisation the intermediate actions are not observable, therefore it seems quite hard to find a context to separate them. So completeness cannot be easily reached and needs much more research effort.

References

1. L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *Intl. Conf. IFIP TCS*, number 1872 in LNCS, pages 333–347. Springer, 2000.
2. L. Cardelli and A. Gordon. Mobile Ambients. In *Proceedings of F0SSaCS'98*, number 1378 in Lecture Notes in Computer Science, pages 140–155. Springer, 1998.
3. L. Cardelli and A. Gordon. Equational properties for Mobile Ambients. In *Proceedings FoSSaCS '99*. Springer LNCS, 1999.
4. L. Cardelli and A. Gordon. A commitment relation for the Ambient Calculus. Available at http://research.microsoft.com/~adg/Publications/ambient-commitment.pdf, 2000.
5. G. Castagna, G. Ghelli, and F. Zappa Nardelli. Typing mobility in the Seal Calculus. In *12th. International Conference on Concurrency Theory*, number 2154 in Lecture Notes in Computer Science, pages 82–101, Springer, 2001.
6. J.C. Godskesen, T. Hildebrandt, and V. Sassone. A calculus of mobile resources. In *CONCUR 2002*, Lecture Notes in Computer Science. Springer, 2002.
7. F. Levi and D. Sangiorgi. Controlling interference in Ambients. In *POPL '00*, pages 352–364. ACM Press, 2000.
8. M. Merro and M. Hennessy. Bisimulation conguences in Safe Ambients. In *Proc. of the 29th ACM Symp. on Principles of Programming Languages*. ACM Press, 2002.
9. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, Parts I and II. *Information and Computation*, 100:1–77, September 1992.
10. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis CST–99–93, University of Edinburgh, 1992.
11. D. Sangiorgi. Bisimulation for Higher-Order Process Calculi. *Information and Computation*, 131(2):141–178, 1996.
12. D. Sangiorgi. Extensionality and intensionality of the ambient logic. In *Proc. of the 28th ACM Symp. on Principles of Programming Languages*, London, 2001.
13. P. Sewell and J. Vitek. Secure composition of insecure components. In *12th IEEE Computer Security Foundations Workshop*, 1999.
14. J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, number 1686 in Lecture Notes in Computer Science, pages 47–77. Springer, 1999.

Composing Strand Spaces

Federico Crazzolara[⋆] and Glynn Winskel

Computer Laboratory University of Cambridge
{fc232,gw104}@cl.cam.ac.uk

Abstract. The strand space model for the analysis of security protocols is known to have some limitations in the patterns of nondeterminism it allows and in the ways in which strand spaces can be composed. Its successful application to a broad range of security protocols may therefore seem surprising. This paper gives a formal explanation of the wide applicability of strand spaces. We start with an extension of strand spaces which permits several operations to be defined in a compositional way, forming a process language for building up strand spaces. We then show, under reasonable conditions how to reduce the extended strand spaces to ones of the traditional kind. For security protocols we are mainly interested in their safety properties. This suggests a strand-space equivalence: two strand spaces are equivalent if and only if they have essentially the same sets of bundles. However this equivalence is not a congruence with respect to the strand-space operations. By extending the notion of bundle we show how to define the strand-space operations directly on "bundle spaces". This leads to a characterisation of the largest congruence within the strand-space equivalence. Finally, we relate strand spaces to event structures, a well known model for concurrency.

1 Introduction

Security protocols describe a way of exchanging data over an untrusted medium so that, for example, data is not leaked and authentication between the participants in the protocol is guaranteed. The last few years have seen the emergence of successful intensional, event-based, approaches to reasoning about security protocols. The methods are concerned with reasoning about the events that a security protocol can perform, and make use of a causal dependency that exists between events. The method of strand spaces [9,10,11] has been designed to support such an event-based style of reasoning and has successfully been applied to a broad number of security protocols.

Strand spaces don't compose readily however, not using traditional process operations at least. Their form doesn't allow prefixing by a single event. Nondeterminism only arises through the choice as to where input comes from, and there is not a recognisable nondeterministic sum of strand spaces. Even an easy definition of parallel composition by juxtaposition is thwarted if "unique origination" is handled as a global condition on the entire strand space. That strand

[⋆] **BRICS** Centre of the Danish National Research Foundation.

spaces are able to tackle a broad class of security protocols may therefore seem surprising. A reason for the adequacy of strand spaces lies in the fact that they can sidestep conflict if there are enough replicated strands available, which is the case for a broad range of security protocols.

This paper has four main objectives. Firstly it extends the strand space formalism to allow several operations on strand spaces to be defined. The operations form a strand-space language. Secondly the wide applicability of strand spaces to numerous security protocols and properties is backed up formally. The paper documents part of the work done in proving the relation between nets and strand spaces we reported in [3]. Thirdly we address another issue of compositionality. We consider languages of strand-space bundles as models of process behaviour and show how to compose such languages so that they may be used directly in giving the semantics of security protocols. Strand spaces that have substantially the same bundles can be regarded as equivalent and are congruent if they exhibit substantially the same *open* bundles. This congruence lays the ground for equational reasoning between strand spaces. Finally we show how strand spaces relate to event structures.

The results in this paper express the adequacy of strand spaces and relate strand spaces to event structures only with respect to the languages, i.e., sets of finite behaviours they generate. This is not unduly restrictive, however, as in security protocols we are mainly interested in safety properties, properties which stand or fall according to whether they hold for all finite behaviours.

2 Strand Spaces

A strand space [11] consists of $\langle s_i \rangle_{i \in I}$, an indexed set of strands. An individual strand s_i, where $i \in I$, is a finite sequence of output or input events carrying output or input actions of the kind $outM$ or inM respectively with M a message built up by encryption and pairing from a set of values (here names) and keys. In the rest of this paper we use n, n_0, n_i to indicate names, A, B, A_0, B_0 to indicate special names which are agent identifiers, and k to stand for a cryptographic key. A name whose first appearance in a strand is on an output message is said to be *originating* on that strand. A name is said to be *uniquely originating* on a strand space if it is originating on only one of its strands.

A strand space has an associated graph whose nodes identify an event of a strand by strand index and position of the event in that strand. Edges are between output and input events concerning the same message and between consecutive events on a same strand. Bundles model protocol runs. A bundle selects those events of a strand space that occur in a run of the protocol and shows the causal dependencies among them which determine the partial order of the events in the run. A bundle is a finite and acyclic subgraph of the strand space graph. Each event in the bundle requires all events that precede it on the same strand (together with the edges that denote the strand precedence). Each input event in the bundle has exactly one incoming edge from an output event.

As an example consider a simplified version of the ISO symmetric key two-pass unilateral authentication protocol (see [2]):

$$\begin{tikzcd}
out\,n_0 \arrow[r] \arrow[d, Rightarrow] & in\,n_0 \arrow[d, Rightarrow] & \dots & out\,n_i \arrow[r] \arrow[d, Rightarrow] & in\,n_i \arrow[d, Rightarrow] & \dots \\
in\,\{n_0, A_0\}_k & out\,\{n_0, A_0\}_k \arrow[l] & & in\,\{n_i, A_0\}_k & out\,\{n_i, A_0\}_k \arrow[l] &
\end{tikzcd}$$

Fig. 1. ISO protocol

$$\begin{tikzcd}
out\,n_0 \arrow[r] \arrow[d, Rightarrow] \arrow[ddr] & in\,n_0 \arrow[d, Rightarrow] & \dots & out\,n_i \arrow[r] \arrow[d, Rightarrow] \arrow[ddr] & in\,n_i \arrow[d, Rightarrow] \\
in\,\{n_0, A_0\}_k & out\,\{n_0, A_0\}_k \arrow[l] & & in\,\{n_i, A_0\}_k & out\,\{n_i, A_0\}_k \arrow[l] \\
out\,n_0 \arrow[r] \arrow[d, Rightarrow] \arrow[uur] & in\,n_0 \arrow[d, Rightarrow] & & out\,n_i \arrow[r] \arrow[d, Rightarrow] \arrow[uur] & in\,n_i \arrow[d, Rightarrow] \\
in\,\{n_0, B_0\}_k & out\,\{n_0, B_0\}_k \arrow[l] & & in\,\{n_i, B_0\}_k & out\,\{n_i, B_0\}_k \arrow[l]
\end{tikzcd}$$

Fig. 2. ISO protocol - symmetric roles

$$\begin{aligned}
&A \to B : n \\
&B \to A : \{n, A\}_k \quad .
\end{aligned}$$

Agents can engage in a protocol exchange under two different roles, the initiator, here A and the responder, here B. In a protocol round the initiator A chooses a fresh name n and sends it to the responder B. After getting the value, B encrypts it together with the initiator's identifier using a common shared key k. After getting the answer to her challenge, A can decrypt using the shared key and check whether the value sent matches the value received. In that case A can conclude that B is in fact operational.

The strand-space graph in Figure 1 describes the simple case of only two agents, A_0 and B_0, acting as initiator and responder respectively. For simplicity the graph has been drawn using the actions labelling the events in place of the events themselves. In this simple case if the strand space had a finite number of strands it would itself form a bundle. In the strand space in Figure 1 all names n_i are uniquely originating – there is only one strand on which the first appearance of n_i is in an output action.

Unique origination intends to describe a name as fresh, perhaps chosen at random, and under the assumptions of Dolev and Yao [4], unguessable. For a construction of parallel composition of strand spaces it is therefore reasonable to require that names uniquely originating on components remain so on the composed strand space. Simple juxtaposition of strand spaces does not ensure this. For example, a strand space for the ISO protocol which allows both agents A_0 and B_0 to engage in the protocol in any of the two possible roles – in Figure 2 the strand space formed out of two copies of the one in Figure 1. Figure 3 shows a possible bundle on such strand space. It describes a protocol run with two complete rounds. One in which A_0 is initiator and B_0 responder, and another where the roles are inverted, though the name n_0 is no longer uniquely originating on that strand space. A name's freshness is with respect to a run of a protocol

$$\begin{tikzcd}
\mathit{out}\, n_0 \arrow[r] \arrow[d, Rightarrow] \arrow[ddr] & \mathit{in}\, n_0 \arrow[d, Rightarrow] \\
\mathit{in}\,\{n_0, A_0\}_k & \mathit{out}\,\{n_0, A_0\}_k \arrow[l] \\
\mathit{out}\, n_0 \arrow[d, Rightarrow] & \mathit{in}\, n_0 \arrow[d, Rightarrow] \\
\mathit{in}\,\{n_0, B_0\}_k & \mathit{out}\,\{n_0, B_0\}_k \arrow[l]
\end{tikzcd}$$

Fig. 3. A bundle for ISO symmetric roles

rather than to the whole set of possible executions. A notion of unique origination "in the bundle" seems more appropriate.

Nondeterminism in strand spaces arises only through the choice in a bundle of where input comes from. There is no way of modelling situations in which bundles may be taken either only over one strand space or over another. Juxtaposing strand spaces as we did for example in Figure 2 allows bundles to include events of both components as is the case for the bundle in Figure 3.

One seems to encounter even more difficulties in the attempt to define a construction of prefixing a strand space with an action. Strands can't branch to parallel sub-strands and prefixing each strand with an action would cause as many repetitions of that action as there are strands participating in a bundle. One could add to a strand space a strand containing only the action meant to prefix the strand space. Actions of the "prefixed" strands, however, would not casually depend on that action.

3 Strand Spaces with Conflict

We extend the definition of strand space, introducing a notion of conflict, which we adapt from event structures [14]. We differ from the original definition of strand spaces in the treatment of unique origination which is taken care of in the definition of bundle rather than being a condition on the entire strand space – the "parametric strand spaces" of [1] achieve a similar effect as to unique origination and are related.

The strands of a strand space are sequences of output and input actions,

$$\mathit{Act} = \{\mathit{out}\,\mathit{new}\,\boldsymbol{n}\,M \mid M \text{ msg.}, \ \boldsymbol{n} \text{ distinct names}\} \cup \{\mathit{in}\,M \mid M \text{ msg.}\} \quad .$$

In $\mathit{out}\,\mathit{new}\,\boldsymbol{n}\,M$, the list $\boldsymbol{n}$ contains distinct names that are intended to be fresh ("uniquely originating") at the event.

Definition 1. A strand space with conflict $(\langle s_i \rangle_{i \in I}, \#)$ consists of:

(i) $\langle s_i \rangle_{i \in I}$ an indexed set of strands with indices I. An individual stand s_i, where $i \in I$, is a finite sequence of output or input actions in Act.
(ii) $\# \subseteq I \times I$ a symmetric, irreflexive binary conflict relation on strand indices.

Strand spaces with an empty conflict relation correspond to those of the standard definition of [11]. We denote by ϵ the empty strand space with no strands and with an empty conflict relation[1]. The empty strand space is different to a strand space $(\langle\lambda\rangle_{i\in I}, \#)$ where each strand is the empty sequence of actions λ. We write $|s|$ for the length of the sequence s.

For a strand space $(\langle s_i\rangle_{i\in I}, \#)$ define the *strand-space graph* $(E, \Rightarrow, \rightarrow, act)$ associated with it as usual (see [11]). It is the graph with nodes (events)

$$E = \{(i,l) \mid i \in I \ , 1 \leq l \leq |s_i|\} \quad ,$$

actions labelling events $act(i,h) = s_i[h]$ and two edge relations. The first expresses precedence among events on the same strand,

$$(i,k) \Rightarrow (i,k+1) \text{ iff } (i,k),(i,k+1) \in E \quad ,$$

and the second expresses all possible communication,

$$(i,l) \rightarrow (j,h) \text{ iff } act(i,l) = out\, new\, \boldsymbol{n}\, M \text{ and } act(j,h) = in\, M \quad .$$

An event is an *input event* if its action is an input action and an event is an *output event* if its action is an output. The names $names(e)$ of an event e are all the names appearing on the action associated with e – the ones that are marked as "new", denoted by $new(e)$, together with those in the message of the action.

Bundles of a strand space describe runs in a computation.

Definition 2. A bundle b of a strand space with conflict $\#$ is a finite subgraph $(E_b, \Rightarrow_b, \rightarrow_b, act_b)$ of the strand-space graph $(E, \Rightarrow, \rightarrow, act)$ such that:

(i) if $e \Rightarrow e'$ and $e' \in E_b$ then $e \Rightarrow_b e'$, *(control precedence)*
(ii) if $e \in E_b$ and $act_b(e) = in\, M$ then there exists a unique $e' \in E_b$ such that $e' \rightarrow_b e$, *(output-input precedence)*
(iii) if $e, e' \in E_b$ such that $act_b(e) = out\, new\, \boldsymbol{n}\, M$ and $n \in \boldsymbol{n} \cap names(e')$ then either $e \Rightarrow_b^* e'$ or there exists an input event e'' such that $n \in names(e'')$ and $e'' \Rightarrow_b^* e'$, *(freshness)*
(iv) if $(i,h),(j,k) \in E_b$ then $\neg(i \# j)$, *(conflict freeness)*
(v) the relation $\Rightarrow_b \cup \rightarrow_b$ is acyclic. *(acyclicity)*

The empty graph, also denoted by λ, is a bundle. It will be clear from the context whether λ stands for the empty bundle or whether it denotes the empty sequence of actions. The empty strand space does not have any bundles.

Points (i), (ii), (v) of the definition of bundle for a strand space with conflict match the standard definition of [11]. Point (iii) ensures freshness of "new" values in a bundle. Point (iv) doesn't allow events from conflicting strands to appear in a bundle. Write $\leq_b$ for the reflexive and transitive closure of $\Rightarrow_b \cup \rightarrow_b$.

Proposition 1. *If b is a bundle then $\leq_b$ is a partial order on E_b.*

[1] We won't make much use of this particular strand space; it is however the identity for the operations of parallel composition and nondeterministic sum of strand spaces.

The relation $\leq_b$ determines the partial order on events occurring in a computation described by b. Names introduced as "new" don't appear on events preceding their introduction and are never introduced as "new" more than once.

Proposition 2. *Let b be a bundle of a strand space and let $e, e' \in E_b$ such that $act_b(e) = out\, new\, \boldsymbol{n}\, M$. If $n \in \boldsymbol{n} \cap names(e')$ then $e \leq_b e'$ and if $act_b(e') = out\, new\, \boldsymbol{m}\, M'$ then $n \notin \boldsymbol{m}$.*

We regard two strand spaces as substantially the same if they differ only on the indices of their strands and so that one strand space can be obtained from the other by a simple "re-indexing" operation[2].

Definition 3. Given $(\langle s_i \rangle_{i \in I}, \#)$ and $(\langle t_j \rangle_{j \in J}, \#')$, two strand spaces, write

$$(\langle s_i \rangle_{i \in I}, \#) \cong (\langle t_j \rangle_{j \in J}, \#')$$

if there exists a bijection $\pi : I \to J$ such that $s_i = t_{\pi(i)}$ for all $i \in I$ and $i \# j$ iff $\pi(i) \#' \pi(j)$ for all $i, j \in I$.

The relation $\cong$ is an equivalence. A bijection π that establishes it is called a *re-indexing of strand spaces.* Let $(\langle s_i \rangle_{i \in I}, \#)$ be a strand space, J be a set, and $\pi : I \to J$ be a bijection. Define the strand space $(\langle t_j \rangle_{j \in J}, \pi(\#))$ where $t_j = s_{\pi^{-1}(j)}$ for all $j \in J$ and $j\, \pi(\#)\, j'$ iff $\pi^{-1}(j) \# \pi^{-1}(j')$ for all $j, j' \in J$. The relation $\pi(\#)$ is irreflexive and symmetric and $(\langle s_{\pi(i)} \rangle_{i \in I}, \pi(\#)) \cong (\langle s_i \rangle_{i \in I}, \#)$.

Proposition 3. *Let $(\langle s_i \rangle_{i \in I}, \#)$ and $(\langle t_j \rangle_{j \in J}, \#')$ be two strand spaces such that $(\langle s_i \rangle_{i \in I}, \#) \cong (\langle t_j \rangle_{j \in J}, \#')$ for a bijection $\pi : I \to J$. If b is a bundle of $(\langle s_i \rangle_{i \in I}, \#)$ then $\pi(b)$ obtained from b by changing all strand indices according to π is a bundle of $(\langle t_j \rangle_{j \in J}, \#')$.*

4 Constructions on Strand Spaces

Prefixing a strand space with an action is complicated by the strand-space formalism not permitting strands to branch. Only if the strand space to be prefixed is such that every two different strands are in conflict can each strand be prefixed with the action. Then the conflict relation disallows repetitions of the prefixing action in bundles. Given $\alpha \in Act$ and a strand space $(\langle s_i \rangle_{i \in I}, \#)$ such that for all $i, j \in I$ if $i \neq j$ then $i \# j$, define

$$\alpha.(\langle s_i \rangle_{i \in I}, \#) \overset{def}{=} (\langle \alpha s_i \rangle_{i \in I}, \#) \quad .$$

We understand the special case of prefixing the empty strand space with an action, to yield the empty strand space $\alpha.\epsilon = \epsilon$. When prefixing a strand space consisting of only empty strands one obtains $\alpha.(\langle \lambda \rangle_{i \in I}, \#) = (\langle \alpha \rangle_{i \in I}, \#)$.

[2] If the indices carry structure (some might involve agent names for example) one might refine the permissible re-indexings.

The *parallel composition* of strand spaces is the disjoint union of their sets of strands and conflict relations. Disjoint union is achieved by tagging each space with a different index. Given a collection of strand spaces $(\langle s_i^k \rangle_{i \in I_k}, \#^k)$ indexed by k in a set K, define

$$||_{k \in K}(\langle s_i^k \rangle_{i \in I_k}, \#^k) \overset{def}{=} (\langle s_h \rangle_{h \in H}, \#)$$

where $H = \sum_{k \in K} I_k$, $s_{(k,i)} = s_i^k$, and where $(k,i) \,\#\, (k',i')$ iff $k = k'$ and $i \,\#^k\, i'$. Two strands are in conflict only if they belong to the same component of the parallel composition and are in conflict within that component. In particular if K is the empty set then the parallel composition yields ϵ.

As a special case of parallel composition of strand spaces consider the strand space obtained by composing infinitely many identical strand spaces. Abbreviate

$$!\,(\langle s_i \rangle_{i \in I}, \#) \overset{def}{=} ||_{k \in \omega}(\langle s_i \rangle_{i \in I}, \#) \quad .$$

One observes that $!\,(\langle s_i \rangle_{i \in I}, \#) = (\langle s_{(n,i)} \rangle_{(n,i) \in \omega \times I}, !\#)$ where $!\#$ is the binary relation over $\omega \times I$ such that $(n,i) \,!\#\, (m,i')$ iff $n = m$ and $i \,\#\, i'$.

The *nondeterministic sum* of strand spaces constructs the same indexed set of strands as the operation of parallel composition. The conflict relation of a summed space however, in addition to the existing conflicts, imposes conflict between every two strands that belong to different components. Given a collection of strand spaces $(\langle s_i^k \rangle_{i \in I_k}, \#^k)$ indexed by k in a set K, define

$$\sum_{k \in K} S_k \overset{def}{=} (\langle s_h \rangle_{h \in H}, \#)$$

where $(k,i) \,\#\, (k',i')$ iff either $k \neq k'$ or ($k = k'$ and $i \,\#^k\, i'$). Two strands are in conflict only if they belong to different components or are already in conflict within a component. The relation $\#$ is irreflexive and symmetric. The indexing set H and the strands remain the same as for parallel composition.

5 A Process Language for Strand Spaces

The language $\mathcal{S}$ of strand spaces has the following grammar:

$$S \ ::= \ L \mid \sum_{j \in J} S_j \mid ||_{j \in J} S_j$$

where $L \in \mathcal{L}$, the language of "sequential strand spaces" is given by

$$L \ ::= \ \langle \lambda \rangle \mid \alpha.L \mid \sum_{j \in J} L_j \quad .$$

The strand space $\langle \lambda \rangle$ has only one strand which is the empty sequence of actions and has empty conflict[3]. The bundles of strand spaces in $\mathcal{L}$ form linearly ordered sets of events, and therefore can be thought of as runs of a sequential process.

[3] Let the index of the empty strand in $\langle \lambda \rangle$ be a distinguished index $*$.

A strand-space term of $\mathcal{S}$ is a "par" process – parallel composition is only at the top level and so the term consists of a parallel composition and sum of sequential processes. The "!-par" processes are those terms of $\mathcal{S}$ of the form $!S$.

6 Open Bundles

The usual semantics of a strand space is expressed in terms of its set of bundles. By broadening to open bundles, the open-bundle space can be built compositionally. An *open bundle* has the structure of a bundle where, however, input events need not necessarily be related to output events. An open bundle is "open" to the environment for communication on input events that are not linked to output events.

Definition 4. An open bundle b of a strand space with conflict $\#$ is a finite subgraph $(E_b, \Rightarrow_b, \rightarrow_b, act_b)$ of the strand-space graph $(E, \Rightarrow, \rightarrow, act)$ such that:

(i) if $e \Rightarrow_G e'$ and $e' \in E_b$ then $e \Rightarrow_b e'$, *(control precedence)*
(ii) if $e' \rightarrow_b e$ and $e'' \rightarrow_b e$ then $e' = e''$, *(output-input correspondence)*
(iii) if $e, e' \in E_b$ s.t. $act(e) = out\,new\boldsymbol{n}M$ and $n \in \boldsymbol{n} \cap names(e')$ then either $e \Rightarrow_b^* e'$ or there exists an input event $e'' \in E_b$ such that $n \in names(e'')$, and $e'' \Rightarrow_b^* e'$, *(freshness)*
(iv) if $(i,h), (j,k) \in E_b$ then $\neg(i \# j)$, *(conflict freeness)*
(v) the relation $\Rightarrow_b \cup \rightarrow_b \cup \hookrightarrow$ is acyclic, where $e \hookrightarrow e'$ if $e \neq e'$ and if $new(e) \cap names(e') \neq \emptyset$. *(acyclicity)*

In an open bundle "freshness" dependencies need not be caught through $\Rightarrow_b$ and $\rightarrow_b$. Point (v) takes account of additional freshness dependencies and excludes graphs like

$$\begin{tikzcd}
in\, n \arrow[d, Rightarrow] & in\, m \arrow[d, Rightarrow] \\
out\, new\, m\; m \arrow[ur] & out\, new\, n\; n
\end{tikzcd}\ .$$

Our constructions for open bundles take us through the intermediary of control graphs (which are similar to pomsets [7] and message sequence charts [5]).

A *control graph*, with indices I, is a graph $(E, \rightarrow, act)$ where $E \subseteq I \times \mathbb{N}$ such that if $(i,h) \in E$ and $h > 1$ then $(i, h-1) \in E$ (when we write $(i,h-1) \Rightarrow (i,h)$), and where $\rightarrow \subseteq E \times E$, and $act : E \rightarrow Act$. Denote by $ind(g)$ the set of indices of a control graph g. Strand-space graphs, bundles and open bundles are examples of control graphs. We say a control graph is an open bundle when it is finite and satisfies axioms (ii), (iii) and (v) above.

Prefixing extends a control graph with a new initial control node $(i,1)$ where i is an index. Every event in the original graph that has index i is shifted one position later in the control structure while keeping its casual dependencies. For i, j indices and $h \in \mathbb{N}$, define $(j,h)/i = (j, h+1)$ if $j = i$ and $(j,h)/i = (j,h)$ otherwise. For an index i, $\alpha \in Act$ and g a control graph, define the control graph

$$(\alpha, i) \,.\, g = (E, \rightarrow, act)$$

where $E = \{(i, 1)\} \cup \{e/i \mid e \in E_g\}$ and $e/i \to e'/i$ whenever $e \to_g e'$. Take $act(i, 1) = \alpha$ and $act(e/i) = act_g(e)$ for every $e \in E_g$.

Control graphs can be *composed* by tagging the number of components to keep them disjoint and juxtaposing them. For i, j indices and $h \in w$, define $j : (i, h) = ((j, i), h)$. For a control graph g and index j define $j : g = (E, \to, act)$ where $E = \{j : e \mid e \in E_g\}$ with $j : e \to j : e'$ whenever $e \to_g e'$ and actions $act(j : e) = act_g(e)$.

The definition of open bundle ensures that b extends to a bundle over a (bigger) strand space. Let g, g' be control graphs. Define $g \preceq g'$ iff $E_g = E_{g'}$, $\to_g \subseteq \to_{g'}$, and $act_g = act_{g'}$. Write $g\!\uparrow = \{b \mid g \preceq b \text{ and } b \text{ an open bundle}\}$.

Lemma 1. *Let b be an open bundle of a strand space S. There exists a strand space T and an open bundle t of T such that among the open bundles in $(t||b)\!\uparrow$ there is a bundle of $T||S$.*

The language of open bundles of a strand-space term is defined as follows:

Definition 5. Let $L \in \mathcal{L}$ and $S \in \mathcal{S}$. Define

$$\mathcal{O}(\langle\lambda\rangle) = \{\lambda\}$$

$$\mathcal{O}(\alpha.L) = \{\lambda\} \;\cup\; \{(\alpha, i)\,.\,\lambda \mid i \in ind(L)\} \;\cup\; \bigcup\{(\alpha, i)\,.\,b\!\uparrow\; \mid b \in \mathcal{O}(L) \setminus \{\lambda\} \;\&\; i \in ind(b)\}$$

$$\mathcal{O}(\sum_{j\in J} S_j) = \{j : b \mid b \in \mathcal{O}(S_j)\}$$

$$\mathcal{O}(||_{j\in J} S_j) = \bigcup\{(\bigcup_{i\in I} i : b_i)\!\uparrow\; \mid I \subseteq J \;\&\; I \text{ finite} \;\&\; \forall i \in I\,.\, b_i \in \mathcal{O}(S_i)\} \quad .$$

Theorem 1. *If S is a strand-space term in $\mathcal{S}$ then the elements of $\mathcal{O}(S)$ are exactly the open bundles of the strand space denoted by S.*

7 Strand Space Equivalence

We have seen an equivalence relation that relates two strand spaces if, via re-indexing, they become the same space. That relation of isomorphism, as expected, is respected by the operations on strand-spaces. When security properties are expressed as safety properties on the language of bundles of a strand space one doesn't, however, want to distinguish between strand spaces that have isomorphic bundle languages.

Let b, b' be two bundles. Write $b \cong b'$ iff there exists a bijection $\phi : E_b \to E_{b'}$ such that

(i) if $e \to_b e'$ then $\phi(e) \to_{b'} \phi(e')$,
(ii) if $e \Rightarrow_b e'$ then $\phi(e) \Rightarrow_{b'} \phi(e')$,
(iii) $act_b(e) = act_{b'}(\phi(e))$.

Definition 6. Let $S, S' \in \mathcal{S}$. Define $\approx$ the symmetric relation such that $S \approx S'$ iff for every bundle b of S there exists a bundle b' of S' such that $b \cong b'$.

Proposition 4. *The relation $\approx$ is an equivalence relation.*

The equivalence relation $\approx$ is not a congruence. For example, the strand-space terms $in\, M.\epsilon$ and $in\, N.\epsilon$ where N and M are two different messages are in the relation $\approx$ – they both have only one bundle, λ. A distinguishing context exists:

$$in\, M.\epsilon \mid\mid out\, M.\epsilon \not\approx in\, N.\epsilon \mid\mid out\, M.\epsilon \quad .$$

The parallel composition on the left hand side has the bundle $in\, M \longleftarrow out\, M$, that on the right only λ.

A context for a strand-space term in the language $\mathcal{S}$ is defined as follows:

$$C ::= [\,] \mid \alpha.C \mid ||_{i\in I} T_i \mid \Sigma_{i\in I} T_i$$

where for each context of the form $||_{i\in I} T_i$ or $\Sigma_{i\in I} T_i$ there is exactly one $i \in I$ such that T_i is a context C and $T_j \in \mathcal{S}$ for all $j \in I \backslash \{i\}$. The context $[\,]$ is a placeholder for a strand-space term. Write $C[S]$ for the term obtained by replacing the strand-space term S for $[\,]$ in context C in the obvious way. An equivalence relation on strand-space terms is a congruence if it respects all contexts.

Definition 7. Let $S, S' \in \mathcal{S}$. Define $\approx_{\mathcal{O}}$ the smallest symmetric relation such that $S \approx_{\mathcal{O}} S'$ if for every open bundle b of S there exists an open bundle b' of S' such that $b \cong b'$.

Theorem 2. *The relation $\approx_{\mathcal{O}}$ is the largest congruence relation inside $\approx$.*

8 Eliminating Conflict

If one doesn't restrict the number of rounds of a protocol an agent can do one can hope to model the protocol with a strand space of the form $!\,(\langle s_i\rangle_{i\in I}, \#)$. In that case a simpler model, obtained by dropping the conflict relation, exhibits substantially the same behaviour as the more complex strand space with conflict.

Lemma 2. *Given I a set of indices, a finite set $A \subseteq \omega \times I$, and $\#$ a conflict relation over I then there exists a bijection $\pi : \omega \times I \to \omega \times I$ where for all $(n, i) \in \omega \times I$ there exists $m \in \omega$ such that $\pi(n, i) = (m, i)$ and $\neg(\pi(u) \,!\#\, \pi(v))$ for all $u, v \in A$.*

The previous lemma and the observation that two different copies of the same strand space have the same strands at corresponding positions, yield:

Theorem 3. *Consider strand spaces $!\,(\langle s_i\rangle_{i\in I}, \emptyset)$ and $!\,(\langle s_i\rangle_{i\in I}, \#)$. Let b be a bundle of $!\,(\langle s_i\rangle_{i\in I}, \emptyset)$. There exists a strand space S such that b is a bundle of S and $S \cong !\,(\langle s_i\rangle_{i\in I}, \#)$.*

The behaviour of a replicated strand space with conflict corresponds, modulo re-indexing, to that of the strand space one obtains by dropping the conflict relation.

Corollary 1. *Consider strand spaces* $!(\langle s_i\rangle_{i\in I}, \emptyset)$ *and* $!(\langle s_i\rangle_{i\in I}, \#)$ *strand spaces.*

(i) If b *is bundle of* $!\,(\langle s_i\rangle_{i\in I}, \#)$ *then* b *is bundle of* $!\,(\langle s_i\rangle_{i\in I}, \emptyset)$.
(ii) If b *is bundle of* $!\,(\langle s_i\rangle_{i\in I}, \emptyset)$ *then there exists a re-indexing* π *such that* $\pi(b)$ *is a bundle of* $!\,(\langle s_i\rangle_{i\in I}, \#)$.

9 Event Structures from Strand Spaces

A bundle is a graph and therefore a set of edges and nodes. It turns out that the bundles of a strand space ordered by inclusion correspond to the finite configurations of a prime event structure. Prime event structures are a particularly simple kind of event structure where the enabling of events can be expressed as a global partial order of causal dependency [6,12].

Definition 8. A prime event structure $E = (E, \#, \leq)$ consists of a set E of events partially ordered by the causal dependency relation $\leq$ and a binary, symmetric, irreflexive conflict relation $\# \subseteq E \times E$, which satisfy

(i) $\{e' \mid e' \leq e\}$ is finite for all $e \in E$, and
(ii) if $e \# e' \leq e''$ then $e \# e''$ for all $e, e', e'' \in E$.

Definition 9. The configurations of $(E, \#, \leq)$ consist of $x \subseteq E$ such that

(i) $\forall e, e' \in x\,.\,\neg(e \# e')$ and *(conflict free)*
(ii) $\forall e, e'.e' \leq e \in x \Rightarrow e' \in x$. *(left closed)*

Write $\mathcal{F}(E)$ for the set of configurations of an event structure and $\mathcal{F}^{fin}(E)$ for its finite configurations. Let $\mathcal{B}$ be the set of bundles of a strand space. A subset X of $\mathcal{B}$ is *compatible* iff there exists $b \in \mathcal{B}$ such that $b' \subseteq b$ for all $b' \in X$. We say X is pairwise compatible if every subset of X with two elements is compatible.

Proposition 5. *The partial order* $(\mathcal{B}, \subseteq)$ *satisfies the following properties:*

(i) if $X \subseteq \mathcal{B}$, X *is finite and pairwise compatible, then* $\bigcup X \in \mathcal{B}$, *(coherence)*
(ii) if $X \subseteq \mathcal{B}$ *and* X *is compatible, then* $\bigcap X \in \mathcal{B}$. *(stability)*

Given $b \in \mathcal{B}$ and $e \in E_b$ define $\lceil e \rceil_b \stackrel{def}{=} \bigcap\{b' \in \mathcal{B} \mid e \in b' \ \wedge\ b' \subseteq b\}$.

Proposition 6. *For every* $b \in \mathcal{B}$ *and every* $e \in E_b$ *the set* $\lceil e \rceil_b$ *is a bundle in* $\mathcal{B}$. *For every finite and compatible* $X \subseteq \mathcal{B}$ *if* $\lceil e \rceil_b \subseteq \bigcup X$ *then* $\exists b' \in X\,.\,\lceil e \rceil_b \subseteq b'$.

We call a bundle $\lceil e \rceil_b$ a prime. The primes form a basis for $(\mathcal{B}, \subseteq)$.

Proposition 7. *Every* $b \in \mathcal{B}$ *can be obtained as* $b = \bigcup\{p \mid p \subseteq b,\ p \textit{ prime}\}$.

Consider the structure $Pr(\mathcal{B}) \overset{def}{=} (P, \#, \subseteq)$ where P is the set of primes of $\mathcal{B}$ and where $p \# p'$ iff the two primes p and p' are not compatible.

Theorem 4. *The structure $Pr(\mathcal{B})$ is a prime event structure.*

Theorem 5. $\phi : (\mathcal{B}, \subseteq) \cong (\mathcal{F}^{fin}Pr(\mathcal{B}), \subseteq)$ *such that* $\phi(b) = \{p \mid p \subseteq b,\ p\ prime\}$ *is an isomorphism of partial orders with inverse map* $\theta : \mathcal{F}^{fin}Pr(\mathcal{B}) \to \mathcal{B}$ *given by* $\theta(x) = \bigcup x$.

Since $Pr(\mathcal{B})$ is a prime event structure the partial order $(\mathcal{F}Pr(\mathcal{B}), \subseteq)$ is a special Scott domain of information, a prime algebraic domain (see [6,12,13,14]). This domain, however, can include configurations which are infinite and therefore are not bundles in the usual sense. The prime event structures $Pr(\mathcal{B})$ underlie the strand-space semantics Syverson gave to the BAN logic [8].

Concluding Remarks. This paper exposes the central position within the theory of strand spaces of the congruence $\approx_{\mathcal{O}}$, based on having the same open bundles. It suggests building an equational theory around $\approx_{\mathcal{O}}$. Its usefulness in supporting abstract specifications would seem to require an operation to hide (or internalise) events in addition to those of Section 5.

References

1. I. Cervesato, N. A. Durgin, P. D. Lincoln, J. C. Mitchell, and A. Scedrov. Relating strands and multiset rewriting for security protocol analysis. *13th CSFW*, 2000.
2. J. Clark and J. Jacob. A survey of authentication protocol literature: V. 1.0. 1997.
3. F. Crazzolara and G. Winskel. Events in security protocols. *8th ACM CCS*, 2001.
4. D. Dolev and A. C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29), 1983.
5. ITU-TS. *ITU-TS Recommendation Z.120: Message Sequence Charts (MSC)*. 1997.
6. M. Nielsen, G. Plotkin, and G. Winskel. Petri nets, Event structures and Domains. *Theoretical Computer Science*, 13, 1981.
7. V. R. Pratt. Modelling concurrency with partial orders. *International Journal of Parallel Programming*, 15(1):33–71, 1986.
8. P. Syverson. Towards a Strand Semantics for Authentication Logic. *Electronic Notes in Theoretical Computer Science*, (20), 1999.
9. J. Thayer and J. Guttman. Authentication tests. *IEEE Symposium on Security and Privacy*, 2000.
10. J. Thayer, J. Herzog, and J. Guttman. Honest ideals on strand spaces. *11th CSFW*, 1998.
11. J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Why is a security protocol correct? *IEEE Symposium on Security and Privacy*, 1998.
12. G. Winskel. *Events in computation.* PhD thesis, University of Edinburgh, 1980.
13. G. Winskel. Event structures. *Adv. Course on Petri nets*, vol. 255 of *LNCS*, 1987.
14. G. Winskel. An introduction to event structures. *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, vol. 354 of *LNCS*, 1988.

Generalising Automaticity to Modal Properties of Finite Structures

A. Dawar[1,⋆] and S. Kreutzer[2]

[1] University of Cambridge Computer Laboratory, Cambridge CB3 0FD, UK
anuj.dawar@cl.cam.ac.uk.
[2] LuFG Mathematische Grundlagen der Informatik, RWTH Aachen
kreutzer@informatik.rwth-aachen.de

Abstract. We introduce a complexity measure of modal properties of finite structures which generalises the automaticity of languages. It is based on graph-automata like devices called labelling systems. We define a measure of the size of a structure that we call *rank*, and show that any modal property of structures can be approximated up to any fixed rank n by a labelling system. The function that takes n to the size of the smallest labelling system doing this is called the *labelling index* of the property. We demonstrate that this is a useful and fine-grained measure of complexity and show that it is especially well suited to characterise the expressive power of modal fixed-point logics. From this we derive several separation results of modal and non-modal fixed-point logics, some of which are already known whereas others are new.

1 Introduction

Modal logics are widely used to express properties of finite (and infinite) state systems for the purpose of automatic verification. In this context, propositional modal logic (also known as Hennessy-Milner logic) is found to be weak in terms of its expressive power and much attention has been devoted to extensions that allow some form of recursion. This may be in the form of path quantifiers as with the branching time temporal logics CTL and CTL* or with a least fixed-point operator as with the μ-calculus. Other extensions have been considered for the purpose of understanding a variety of fixed-point operators or classifying their complexity. Examples include L_μ^ω, the higher dimensional μ-calculus introduced by Otto [6], and MIC, the modal iteration calculus, introduced in [3]. The former was introduced specifically to demonstrate a logic that exactly characterises the polynomial-time decidable bisimulation invariant properties of finite-state systems, while the latter was studied in an investigation into the difference between least and inflationary fixed points.

The study of these various extensions of propositional modal logic has thrown up a variety of techniques for analysing their expressive power. One can often show that one logic is at least as expressive as another by means of an explicit translation of formulae of the first into the second. Establishing separations between logics is, in general, more involved. This requires identifying a property expressible in one logic and

⋆ Research supported by EPSRC grant GR/N23028.

proving that it is not expressible in the other. Many specialised techniques have been deployed for such proofs of inexpressibility, including diagonalisation, bisimulation and other Ehrenfeucht-Fraïssé style games, complexity hierarchies and automata-based methods such as the pumping lemma.

In this paper, we introduce an alternative complexity measure for modal properties of finite structures which we call the *labelling index* of the property and demonstrate its usefulness in analysing the expressive power of modal fixed-point logics. The labelling index generalises the notion of the automaticity of languages (see [7].) The automaticity of a language (i.e. set of strings) L is the function that maps n to the size of the least deterministic finite automaton which agrees with L on all strings of length n or less. We generalise this notion in two steps, first studying it for classes of finite trees and then for classes of finite, possibly cyclic, transition systems.

We introduce automata-like devices called labelling systems and a measure on finite structures that we call *rank*. We show that any modal property of finite structures (or equivalently, any class of finite structures closed under bisimulation) can be approximated up to any fixed rank n by a labelling system. The function that takes n to the size of the smallest labelling system that does this is the labelling index of the property. We demonstrate that this is a useful and fine-grained measure of the complexity of modal properties by deriving a number of separation results using it, including some that were previously known and some that are new. We show that any property that is definable in propositional modal logic has constant labelling index. In contrast, any property that is definable in the μ-calculus has polynomial labelling index and moreover, there are properties definable in L_μ whose labelling indices have a linear lower bound. Similarly we obtain exponential upper and lower bounds on the labelling index of properties definable in MIC. We demonstrate that MIC is not the bisimulation-invariant fragment of monadic IFP. We also investigate the relationship between labelling index and conventional time and space based notions of complexity. Finally, we investigate the labelling index of the trace equivalence problem over specific classes of structures and deduce interesting results about its expressibility in various fixed-point logics.

Due to lack of space, proofs of the results are only sketched.

2 Background

In this section, we give a brief introduction to modal logic and its various fixed-point extensions. A detailed study of these logics can be found in [2,1,3].

Propositional Modal Logic. For the rest of the paper fix a set $\mathcal{A}$ of actions and a set $\mathcal{P}$ of atomic propositions. Modal logics are interpreted on *transition systems*, also called *Kripke structures*, which are edge and node labelled graphs. The labels of the edges come from the set $\mathcal{A}$ of actions, whereas the nodes are labelled by sets of propositions from $\mathcal{P}$.

Modal logic (ML) is built up from atomic propositions $p \in \mathcal{P}$ using boolean connectives and the *next-modalities* $\langle a \rangle$, $[a]$ for each $a \in \mathcal{A}$. Formulae $\varphi \in \mathrm{ML}$ are always evaluated at a particular node in a transition system. We write $\mathcal{K}, v \models \varphi$ if φ holds at the node v in the transition system $\mathcal{K}$. The semantics of ML-formulae is as usual with $\mathcal{K}, v \models \langle a\rangle\varphi$ if there is an a-successor u of v such that $\mathcal{K}, u \models \varphi$ and, dually, $\mathcal{K}, v \models [a]\varphi$ if for all a-successors u of v, $\mathcal{K}, u \models \varphi$.

Bisimulations. Bisimulation is a notion of behavioural equivalence for transition systems (see, e.g. [8] for a definition). Modal logics, like ML, CTL, the μ-calculus etc. do not distinguish between transition systems that are bisimulation equivalent. We write $\mathcal{K}, v \sim \mathcal{K}', v'$ to denote that the two transition systems are equivalent by bisimulation.

For a transition system $\mathcal{K}$ we write $\mathcal{K}_{/\sim}$ for its quotient under bisimulation. That is, $\mathcal{K}_{/\sim}$ is the transition system whose states are the equivalence classes of states of $\mathcal{K}$ under bisimulation and, if $[v]$ denotes the equivalence class containing v, then $[v] \in [\![p]\!]^{\mathcal{K}/\sim}$ if $v \in [\![p]\!]^{\mathcal{K}}$ and there is an a-transition from $[u]$ to $[v]$ in $\mathcal{K}_{/\sim}$ if, and only if, there is an a-transition from u to v in $\mathcal{K}$. It is easily verified that $\mathcal{K}, v \sim \mathcal{K}_{/\sim}, [v]$.

Modal Fixed-Point Logics. We now consider two fixed-point extensions of modal logic: the modal μ-calculus and the modal iteration calculus (MIC).

Syntactically they are defined as the closure of modal logic under the following formula building rules. Let $\varphi_1, \dots, \varphi_k$ be formulae with free proposition symbols $X_1, \dots, X_k$ and let $S := \{X_1 \leftarrow \varphi_1, \dots, X_k \leftarrow \varphi_k\}$ be a system of rules. Then $\mu X_i : S$ and $\nu X_i : S$ are formulae of L_μ and $(\textbf{ifp}\ X_i : S)$ is a formula of MIC, where, in the case of L_μ, the rule is restricted to systems S where all formulae φ_i in S are positive in all fixed-point variables X_j.

On any finite transition system $\mathcal{K}$ with universe V, such a system S of rules defines an operator F_S taking a sequence $(X_1, \dots, X_k)$ of subsets of V to the sequence $(F_S(X_1), \dots, F_S(X_k))$, where $F_S(X_i) := \{u : (\mathcal{K}, (X_i)_{1\leq i\leq k}), u \models \varphi_i\}$. This operator, again, inductively defines for each finite ordinal α, a sequence of sets $(X_1^\alpha, \dots, X_k^\alpha)$ as follows. For all i, $X_i^0 := \emptyset$ and for $0 < \alpha < \omega$, $X_i^\alpha := (F_S(\overline{X}^{\alpha-1}))_i$.

As the formulae in the μ-calculus are required to be positive in their free fixed-point variables, the operator F_S induced by a system of L_μ-operators is monotone and thus always has a least and a greatest fixed point. By a well known result of Knaster and Tarski, the least fixed point is also reached as the fixed point $(X_1^\infty, \dots, X_k^\infty)$ of the sequence of stages as defined above, and the greatest fixed point is reached as the limit of a similar sequence of stages, where the induction is not started with the empty set but with the entire universe, i.e. $X_i^0 := V$. The semantics of a formula $\mu X_i : S$ is defined as $\mathcal{K}, u \models \mu X_i : S$ if, and only if, u occurs in the i-th component of the least fixed point of F_S if, and only if, $u \in X_i^\infty$. Analogously, $\mathcal{K}, u \models \nu X_i : S$ if, and only if, u occurs in the i-th component of the greatest fixed point of F_S [1].

The next fixed-point extension of ML we consider is the modal iteration calculus introduced in [3]. It is designed to overcome the restriction of L_μ to positive formulae, but still guarantee the existence of a meaningful fixed point. This is achieved by taking at each induction step the union with the previous stage, i.e. $X_i^{\alpha+1}$ is defined as $X_i^{\alpha+1} := X_i^\alpha \cup (F_S(\overline{X}^\alpha))_i$. Thus, the stages of the induction are increasing and lead to a fixed point $(X_1^\infty, \dots, X_k^\infty)$. Again, $\mathcal{K}, u \models \textbf{ifp}\ X_i : S$ if, and only if, $u \in X_i^\infty$.

[1] In most presentations of the μ-calculus simultaneous inductions are not considered. Nothing is lost by such a restriction as the least fixed point of a system S can also be obtained by nested fixed points of simple inductions (see [1]).

Another fixed-point extension of modal logic that we consider is L_μ^ω, the higher-dimensional μ-calculus defined by Otto. We refer the reader to [6] for a precise definition. Here we only note that this logic permits the formation of least fixed points of positive formulae φ defining not a set X, but a relation X of any arity. Otto shows that, restricted to finite structures, this logic can express exactly the bisimulation-closed properties that are polynomial-time decidable.

It is immediate from the definitions that, in terms of expressive power, we have $\mathrm{ML} \subseteq L_\mu \subseteq \mathrm{MIC} \subseteq \mathrm{IFP}$, where IFP denotes the extension of first-order logic by inflationary fixed points. As IFP is equivalent to least fixed-point logic (LFP) and L_μ^ω is the bisimulation invariant fragment of LFP, is follows that $\mathrm{MIC} \subseteq L_\mu^\omega$. Indeed, all of these inclusions are proper. The separations of MIC from L_μ and L_μ^ω were shown in [3]. The analysis of the labelling index of properties expressible in the logics provides a uniform framework for both separations.

There is a natural translation of L_μ formulae into monadic second-order logic. Indeed, Janin and Walukiewicz [5] show that a formula of monadic second-order logic is bisimulation invariant if, and only if, it is equivalent to a formula of L_μ. Thus, the separation of MIC from L_μ shows that MIC can express properties that are not definable in monadic second-order logic. In [3], the question was posed whether MIC could be characterised as the bisimulation invariant fragment of any natural logic. The most natural candidate for this appears to be the monadic fragment of IFP—the extension of first order predicate logic with inflationary fixed points. However, by an analysis of the labelling index of properties definable in this logic, we show that it can express bisimulation-invariant properties that are not in MIC.

3 Automaticity on Strings and Trees

The automaticity of a language $L \subseteq \Sigma^*$ is the function that maps n to the size of the minimal deterministic automaton that agrees with L on all strings of length at most n. This function is constant if, and only if, L is regular and is at most exponential for any language L.

In [3] it was shown that MIC is strictly less expressive than L_μ^ω. The full version of that paper makes it clear that the method used to separate the logics is a generalisation of the definition of automaticity from string languages to classes of finite trees, closed under bisimulation. Automata that operate on trees have been widely studied in the literature (see, for instance, [4]). We consider "bottom-up" automata that have the property that the class of trees accepted is necessarily closed under bisimulation. Formally, a bottom-up tree automaton is $\mathcal{A} = (Q, A, \delta, F, s)$, where $s \in Q$ is a start state, and $\delta = 2^{Q \times A} \to Q$. We say such an automaton accepts a tree $\mathcal{T}$, if there is a labelling $l : \mathcal{T} \to Q$ of the nodes of $\mathcal{T}$ such that for every leaf v, $l(v) = s$, the root of $\mathcal{T}$ is labelled $q \in F$, and $l(v) = \delta(\{(l(w), a) : v \xrightarrow{a} w\})$. We have, for simplicity, assumed that $\mathcal{T}$ is a transition system where the set of propositions $\mathcal{P}$ is empty. The automata are easily generalised to the case where such propositions are present. Indeed the labelling systems we introduce in Definition 4.6 below offer such a generalisation.

For a bisimulation-closed class $\mathcal{C}$ of trees, its automaticity can be defined (see the full version of [3]) as the function mapping n to the smallest bottom-up tree automaton agreeing with $\mathcal{C}$ on all trees of *height* n. Height is the appropriate measure to use on a tree since it bounds the number of steps the automaton takes. This version of automaticity was used in particular to separate the expressive power of MIC from that of L^ω_μ. Indeed, one can establish the following facts about the automaticity of classes of trees definable in modal fixed-point logics.

Proposition 3.1. *1. Every class of trees definable in L_μ has constant automaticity.*
2. Every class of trees definable in MIC *has at most exponential automaticity.*
3. There is a class of strings definable in MIC *that has exponential automaticity.*
4. There is a class of trees definable in L^ω_μ that has non-elementary automaticity.

Statement (1) follows from the fact that for any formula φ of L_μ we can construct a bottom-up tree automaton which accepts exactly those trees that satisfy φ (see [11]). Statements (2), (3) and (4) are shown in [3]. However, (2) can also be derived as a special case of Theorem 5.1 proved below. The particular class of trees used to establish (4) is the *bisimulation problem*. This is the class of trees T such that for any subtrees T_1 and T_2 rooted at children of the root of T, we have $T_1 \sim T_2$. It can be seen that the automaticity of this class is the maximum possible.

Monadic Inflationary Fixed-Point Logic. We now look at the automaticity of the bisimulation-invariant fragment of monadic IFP on trees and show that there is no elementary lower bound for it. A consequence is that MIC is not the bisimulation invariant fragment of monadic IFP, something that could naturally be conjectured, given that the μ-calculus is the bisimulation-invariant fragment of monadic least fixed-point logic.

We first introduce monadic inflationary fixed-point logic (M-IFP) as the closure of first-order logic under the following rule. If $\varphi(X, x)$ is a formula with a free unary relational variable X and a free first-order variable x, then for any term t, $[\mathbf{ifp}_{X,x}\, \varphi](t)$ is also a formula. The semantics is defined as for MIC, i.e. $[\mathbf{ifp}_{X,x}\, \varphi]$ defines the inflationary fixed point of the operator induced by φ.

The properties we are going to construct that are definable in M-IFP and have high automaticity are based on the use of trees to encode sets of integers in a number of ways of increasing complexity. To be precise, for each natural number k, we inductively define an equivalence relation $\simeq_k$ on trees as follows.

Definition 3.2. *For any two trees t and s, write $t \simeq_0 s$ just in case t and s have the same height and $t \simeq_{k+1} s$ just in case the set of $\simeq_k$-equivalence classes of the subtrees rooted at the children of the root of t is the same as the set of $\simeq_k$-equivalence classes of the subtrees rooted at the children of the root of s.*

By abuse of notation, we will also think of these relations as relations on the nodes of a tree $\mathcal{T}$. In this case, by $u \simeq_k v$ we mean $t_u \simeq t_v$ where t_u and t_v are the trees rooted at u and v respectively. A simple induction establishes the following lemma.

Lemma 3.3. *The number of distinct $\simeq_k$ equivalence classes of trees of height $n + k$ or less is k-fold exponential in n.*

Now, let $\mathcal{C}_k$ be the class of trees $\mathcal{T}, v$ with root v such that all successors of the root are $\simeq_k$-equivalent. By Lemma 3.3, the automaticity of $\mathcal{C}_k$ is at least k-fold exponential. Also it is easy to see that $\simeq_k$-equivalence is M-IFP-definable. This establishes the following theorem.

Theorem 3.4. *For every elementary function f, there is a property with automaticity $\Omega(f)$ definable in* M-IFP.

It follows from this that there are bisimulation invariant properties definable in M-IFP that are not definable in MIC. This contrasts with L_μ whose expressive power coincides precisely with the bisimulation invariant fragment of monadic LFP. This result dashes hopes of characterising MIC as the bisimulation-invariant fragment of a natural predicate logic, a question that was posed in [3].

Corollary 3.5. MIC *is strictly contained in the bisimulation invariant fragment of* M-IFP.

4 Labelling Index

We now generalise automaticity further to finite transition systems that are not necessarily acyclic. This necessitates some changes. First, we have to extend the automata model to devices operating on arbitrary finite transition systems. As the structures may have cycles, there is no natural starting or ending point for an automaton. For this reason, we have refrained from calling the devices automata and adopted the term *labelling systems* instead. The systems are deterministic in that the label attached to a node is completely determined by the labels at its successors and the propositions that hold at the node. In this sense, the devices are also bottom-up. The formal definition is given in Definition 4.6.

However, in order to have a meaningful measure of the growth rate of these devices, we require a measure of the size of finite transitions systems that generalises the length of a string and the height of a tree. We proceed to this first.

Definition 4.1. *The* rank *of a structure $\mathcal{K}, v$ is the largest n such that there is a sequence of* distinct *nodes $v_1, \dots, v_n$ in $\mathcal{K}$ with $v = v_1$ and there is a path from v_i to v_{i+1} for each i.*

It is easy to see that the rank of a tree is indeed its height (taking the height of a tree with a single node as being 1) and the rank of any acyclic structure is equal to the length of the longest non-repeating path. This observation can be further generalised by the following equivalent characterisation of rank.

Definition 4.2. *The* block decomposition *of a structure $\mathcal{K}$ is the acyclic graph $G = (V, E)$ whose nodes are the strongly connected components of $\mathcal{K}$ and $(s, t) \in E$ if, and only if, for some $u \in s$ and some $v \in t$, there is an action a such that $u \xrightarrow{a} v$. For each node s of G, we write $weight(s)$ for the number of nodes u of $\mathcal{K}$ such that $u \in s$. The rank of a node s of G is defined inductively by $rank(s) = weight(s) + \max\{rank(t) : (s, t) \in E\}$.*

The block rank *of a rooted finite transition system $\mathcal{K}, v$ is defined as the rank of the block containing v in the block decomposition of $\mathcal{K}$.*

Lemma 4.3. *The block rank of $\mathcal{K}, v$ is equal to its rank.*

When relating tree-automata to fixed-point logics as in Proposition 3.1, the key property of the height of the tree is that it bounds the length of any simple fixed point induction that can be defined in L_μ or MIC. We show that this carries over to our definition of rank.

Lemma 4.4. *If $\varphi(X)$ is a formula of* MIC*, and n is the rank of $\mathcal{K}, v$, then $\mathcal{K}, v \models$ $(\mathbf{ifp}\ X : \varphi)$ if, and only if, $v \in X^n$.*

Proof. The proof is by induction on n. The basis, $n = 1$ is trivial. For the induction step, we show that for $k \geq 1$, if $v \in X^{k+1}$ but $\boldsymbol{v} \notin X^k$, then there must be a u reachable from v such that $u \in X^k \setminus X^{k-1}$. □

While the rank of a structure $\mathcal{K}, v$ provides a combinatorial measure that bounds the closure ordinals of simple inductions, it is not an exact characterisation. Nor can we expect it to be exact because it is clear that the closure ordinals are invariant under bisimulation while rank is not. It may be more appropriate therefore to consider the rank, not of a structure $\mathcal{K}$, but of its quotient under bisimulation $\mathcal{K}_{/\sim}$. With this, we do indeed get the required converse to Lemma 4.4.

Lemma 4.5. *If the rank of $\mathcal{K}_{/\sim}$ is n, there is a formula $\varphi(X)$, positive in X, whose closure ordinal on $\mathcal{K}$ is n.*

An immediate consequence of Lemmas 4.4 and 4.5 is that the maximal closure ordinals of simple MIC and simple L_μ formulae on any structure are the same.

We are now ready to introduce labelling systems, which generalise bottom-up tree automata to transition systems that are not necessarily acyclic.

Definition 4.6. *A* labelling system *$\mathcal{L}$ is a quintuple $\mathcal{L} := (Q, \mathcal{A}, \mathcal{P}, \delta, \mathcal{F})$, where Q is a finite set of labels, $\mathcal{A}$ a finite set of actions, $\mathcal{P}$ a finite set of proposition symbols, $\mathcal{F} \subseteq Q$ a set of* accepting labels*, and δ a total function $\delta : 2^{Q \times \mathcal{A}} \times 2^{\mathcal{P}} \to Q$, the* transition function*.*

For every Kripke-structure $\mathcal{K} := (V, (E_a)_{a \in \mathcal{A}}, (P_i)_{i \in \mathcal{P}})$ and node $v \in V$, the labelling system $\mathcal{L}$ accepts $\mathcal{K}, v$, denoted $\mathcal{K}, v \models \mathcal{L}$, if, and only if, there is a function $f : V \to Q$ such that for each $s \in V$, $f(s) = \delta(\{(f(s'), a) : a \in \mathcal{A}$ and $(s, s') \in E_a\}, \{i : i \in \mathcal{P}$ and $s \in P_i\})$, and $f(v) \in \mathcal{F}$.

As δ is functional, labelling systems are deterministic devices. Indeed, on well-founded trees, labelling systems and bottom-up tree automata are equivalent. On the other hand, if the structures may contain cycles, some form of nondeterminism is present as acceptance is defined in terms of the *existence* of a labelling. Thus, for a given structure and a given labelling system, there may be more than one labelling function f that witnesses the fact that $\mathcal{L}$ accepts $\mathcal{K}, v$.

The class of structures accepted by a labelling system is not necessarily closed under bisimulation. This can be seen in the following simple example.

Example 4.7. Consider the labelling system $\mathcal{L} = (Q, A, P, \delta, \mathcal{F})$ given by: $Q = \{q, q'\}$, $A = \{a\}$, $P = \emptyset$, $\mathcal{F} = \{q\}$ and where δ is given by the rules $\delta(\emptyset) = q$, $\delta(\{(q, a)\}) = q'$, $\delta(\{q', a\}) = q$ and $\delta(\{(q, a), (q', a)\}) = q$, where we have dropped the second argument to δ as it is always $\emptyset$.

This labelling system accepts a simple cycle if, and only if, it is of even length.

As we are especially interested in labelling systems that define bisimulation-closed classes of structures, we consider the following definition.

Definition 4.8. *A labelling system $\mathcal{L}$ is* $\sim$-consistent*, if for all Kripke-structures $\mathcal{K}, v$, whenever $\mathcal{K}, v \models \mathcal{L}$ then there is a labelling f witnessing this and for all s, s', $\mathcal{K}, s \sim \mathcal{K}, s'$ implies $f(s) = f(s')$.*

It might seem that a more natural condition would be obtained just by requiring the class of structures defined by $\mathcal{L}$ to be closed under bisimulation, as in the following definition.

Definition 4.9. *A labelling system $\mathcal{L}$ is* $\sim$-invariant *if, whenever $\mathcal{K}, v \models \mathcal{L}$ and $\mathcal{K}, v \sim \mathcal{K}', v'$ then $\mathcal{K}', v' \models \mathcal{L}$.*

As it happens, these two definitions are equivalent for the structures that are of interest to us. Call $\mathcal{K}, v$ *connected* if, for every node u, there is a path from v to u in $\mathcal{K}$.

Lemma 4.10. *On connected structures, a labelling system is $\sim$-consistent if, and only if, it is $\sim$-invariant.*

While any $\sim$-consistent labelling systems $\mathcal{L}$ defines a class of $\sim$-invariant structures, not every bisimulation-closed class $\mathcal{C}$ of structures is given by such a labelling system. However, as we see below, $\mathcal{C}$ is defined by a family of systems. In order to define the family we use the rank of a structure as a measure of its size.

We show now that every bisimulation closed class of transition systems can be accepted by a family of labelling systems as follows. For this, note that the rank is trivially bounded by the size of the transition system and that there are only finitely many bisimulation equivalence classes of structures of a given rank n. Taking a state for each such class yields the desired labelling system.

Lemma 4.11. *Let $\mathcal{C}$ be a bisimulation closed class of finite structures. For each n there is a $\sim$-consistent labelling system $\mathcal{L}_n$ such that for any structure $\mathfrak{A}$ with rank$(\mathfrak{A}) \leq n$, $\mathcal{L}_n$ accepts $\mathfrak{A}$ if, and only if, $\mathfrak{A} \in \mathcal{C}$.*

The minimal size in terms of n of the labelling systems in a family such as that in Lemma 4.11 can be seen as a measure of the complexity of the class $\mathcal{C}$. This leads to the definition of the labelling index of classes of transition systems, which generalises the automaticity of languages and classes of trees.

Definition 4.12. *Let $\mathcal{C}$ be a bisimulation closed class of finite structures. The* labelling index *of $\mathcal{C}$ is defined as the function $f : n \mapsto |\mathcal{L}_n|$ mapping natural numbers n to the number of labels of the smallest labelling system such that for any $\mathcal{K}, v$ of rank n or less, $(\mathcal{K}, v) \in \mathcal{C}$ if, and only if, $\mathcal{K}, v \models \mathcal{L}_n$.*

A comparison of labelling systems with other automata models on graphs, such as tiling systems [9,10] is instructive. Significant differences are that tiling systems are generally nondeterministic and the label attached to a node depends on its predecessors as well as its successors.

5 Labelling Indices of Modal Logics

In this section, we aim to establish upper and lower bounds on the labelling index of classes of structures definable in modal logics such as ML and its various fixed-point extensions.

The modal iteration calculus. It was shown by Dawar, Grädel, and Kreutzer in [3] that any class of trees definable in MIC has at most exponential automaticity. The proof translates easily to the labelling index on arbitrary structures, as sketched below.

Let φ be a formula in MIC and let Φ be the set of sub-formulae of φ. Further let $X_1, \dots, X_k$ be the fixed-point variables occurring in φ. Clearly, for every transition system $\mathcal{K}, v$ of rank n the fixed point of each inflationary induction must be reached after at most n stages.

For every transition system $\mathcal{K}, v$ of rank n we define the *φ-type of $\mathcal{K}, v$* as the function $f : \{0, \dots, k \cdot n\}^k \to 2^{\Phi}$ such that $\psi \in \Phi$ occurs in $f(\bar{\imath})$ if, and only if, ψ holds at the root v of $\mathcal{K}$ if the variables occurring free in ψ are interpreted by the stages $X_j^{i_j}$. A function $f : \{0, \dots, k \cdot n\}^k \to 2^{\Phi}$ is a *φ-type* if it is a φ-type of a transition system.

We are able to define for each formula in MIC a family of labelling systems accepting the class of its models, where the φ-types serve as labels. This gives us the following theorem.

Theorem 5.1. *Every* MIC *definable class of transition systems has at most exponential labelling index.*

There is a corresponding lower bound, as it is shown in [3] that there are MIC-definable classes of structures with exponential labelling index.

Another corollary of the results refers to modal logic. As ML-formulae can be seen as MIC-formulae without any fixed-point operators, the number of φ-types for a ML-formula φ depends only on φ and is therefore constant. Thus we immediately get the following.

Corollary 5.2. *Every property definable in* ML *has constant labelling index.*

The modal μ-calculus. The main difference between the argument for MIC considered above and L_μ is monotonicity. This has a major impact on the definition of labelling systems accepting L_μ definable classes of structures. Consider the labelling systems as defined for MIC-formulae φ. In each node u of the structures we remembered for the sub-formulae of φ every tuple $(i_1, \dots, i_k)$ of induction stages where the sub-formula becomes true at u. As L_μ formulae are monotone, if a sub-formula is true at a tuple of stages $(i_1, \dots, i_k)$ it will also be true at all higher stages of μ and all lower stages of ν-operators. Thus, if we only had one fixed-point operator, say μX, it would suffice to mark each node u of the structure by the number of the stage at which it is included into the fixed point of X, or to give it a special label if it is not included at all. We would thus only have linearly many labels in the labelling system.

But monotonicity also helps if there are more than one fixed-point operator. The reason is that if a formula is true at a node u and a tuple of stages $\bar{\imath}$, then it is also true at u if all or some of its free fixed-point variables are interpreted by their respective fixed points. With this, it turns out to be sufficient to consider in each node u of the transition system only those tuples $\bar{\imath}$ of stages where at most one fixed-point induction has not reached its fixed point. As there are only polynomially many such tuples we get a polynomial upper bound on the size of the labelling systems. We omit a detailed proof for lack of space.

Theorem 5.3. *Every class of transition systems definable in L_μ has at most polynomial labelling index.*

A consequence of the proof is that if a L_μ-formula does not use any μ-operators, the class of structures defined by it has constant labelling index. Thus, to give an example of a L_μ definable class of structures with non-constant labelling index, the exclusive use of ν-operators is not sufficient. But it can easily be seen, using pumping arguments, that to express reachability, constant size labelling systems are not sufficient.

Lemma 5.4. *There is a L_μ-definable class $\mathcal{C}$ of structures that has a linear lower bound on its labelling index.*

Proof. Let $\mathcal{C}$ be the class of transition systems such that there is a node reachable from the root labelled by the proposition p. Obviously, $\mathcal{C}$ can be accepted by a family of labelling systems with linear growth function.

On the other hand, each family of labelling systems accepting $\mathcal{C}$ must have at least linear size. Assume otherwise and suppose that for some $n > 2$ there is a labelling system $\mathcal{L}$ of size less than n accepting the class $\mathcal{C}_n$ of structures from $\mathcal{C}$ of rank at most n. Consider the structure $\mathcal{K} := (\{0, \dots n-1\}, E, P)$ with $E := \{(i, i+1) : 0 \leq i < n-1\}$ and $P := \{n-1\}$. Obviously $\mathcal{K}, 0 \in \mathcal{C}_n$ and thus $\mathcal{K}, 0$ is accepted by $\mathcal{L}$. As there are less than n labels, there must be two different nodes $u < v$ in $\mathcal{K}$ labelled by the same label q in $\mathcal{L}$. But then the same labelling by label from $\mathcal{L}$ also witnesses that the system $\mathcal{K}' := (\{0, \dots, v\}, E', P')$ where $E' := \{(i, i+1) : 0 \leq i < v\} \cup \{(v, u+1)\}$ and $P' := \emptyset$, would be accepted by $\mathcal{L}$. As $\mathcal{K}', 0 \notin \mathcal{C}$ we get a contradiction. □

Thus, the μ-calculus has a polynomial labelling index in the sense that every L_μ definable property has polynomial labelling index and there are L_μ-definable properties with a linear lower bound on the labelling index.

This also shows that various ML extensions like LTL, CTL, or CTL* have non-constant labelling index, as they can express reachability.

6 Labelling Index and Complexity

We begin by contrasting labelling index with the usual notion of computational complexity in terms of machine time measured as a function of the size of the structure. We demonstrate that the two measures are not really comparable by exhibiting a class of structures that is decidable in polynomial time but has non-elementary labelling index and on the other hand an NP-complete problem that has exponential labelling index.

The first of these is the class of finite trees $\mathcal{F}$ such that if t_u are t_v are subtrees rooted at a successor of the root, then $t_u \sim t_v$. As was shown in [3], there is no elementary bound on the automaticity of this class, but it is decidable in time polynomial in the *size* of the tree. This yields the following result.

Proposition 6.1. *There is a polynomial-time decidable class of Kripke structures with non-elementary labelling index.*

In contrast, we can construct an NP-complete problem of much lower labelling index. We obtain this by encoding propositional satisfiability as a class of structures $\mathcal{S}$

closed under bisimulation, and demonstrate that it is accepted by an exponential family of labelling systems.

Theorem 6.2. *There are* NP*-complete classes with exponential labelling index.*

It is an open question whether the exponential bound in Theorem 6.2 can be lowered.

The trace-equivalence problem. We now apply our methods to a particular problem that is of interest from the point of view of verification—the *trace-equivalence* problem. We determine exactly the labelling index of a number of variations of the problem and thereby derive results about their expressibility in various modal fixed-point logics.

Consider a Kripke structure $\mathcal{K}, v$ with set of actions $\mathcal{A}$ and a distinguished proposition symbol $\mathcal{F}$ denoting accepting nodes. We define the set of *traces* of the structures to be the set $\mathcal{T} \subseteq \mathcal{A}^*$ such that $t \in \mathcal{T}$ just in case there is a path labelled t from v to a node in $\mathcal{F}$. Two structures are said to be *trace equivalent* if they have the same set of traces.

To define the decision problem of trace equivalence as a bisimulation-closed class of structures, we consider $\mathcal{E} = \{\mathcal{K}, v :$ if $v \to u$ and $v \to w$ then $\mathcal{K}, u$ and $\mathcal{K}, w$ are trace equivalent$\}$. The unary trace-equivalence problem is $\mathcal{E}$ restricted to structures over a vocabulary with a single action, i.e. $\mathcal{A} = \{a\}$. Similarly, we define binary trace equivalence to be the class of structures over a vocabulary with action set $\{a, b\}$ that are also in $\mathcal{E}$.

Theorem 6.3. *(i) On acyclic structures, unary trace equivalence has exponential labelling index.*
(ii) On acyclic structures, binary trace equivalence has double exponential labelling index.
(iii) On arbitrary structures, unary trace equivalence has double exponential labelling index.
(iv) On arbitrary structures, binary trace equivalence has a treble exponential labelling index.

Proof. The proofs of the four statements are fairly similar, and we establish them all at once. In each case, we identify with each node v in a structure a *discriminating set* D_v which is a finite set of traces available from v such that if u and v are not trace equivalent then $D_u \neq D_v$. It is easy to see that if two nodes in an acyclic structure of rank n are trace inequivalent, then there is a trace of length at most n that distinguishes them. It can also be shown that in an arbitrary structure of rank n, two inequivalent nodes are distinguished by a trace of length at most 2^n. Thus, if $\mathcal{D}$ is the set of all discriminating sets, $\mathcal{D}$ is exponential in n in case 1; double exponential in cases 2 and 3; and treble exponential in case 4. This allows us to construct the labelling systems establishing the upper bounds.

For the lower bounds, note that, in the case of acyclic structures, for every set $D \in \mathcal{D}$ we can easily construct a rooted structure $\mathcal{K}_D, v$ of rank n such that its discriminating set is exactly D. For the case of structures with cycles, the construction is not as straightforward but, it can be shown that there is a polynomial p such that there is a collection of 2^{2^n} unary structures (and $2^{2^{2^n}}$ binary structures) of rank $p(n)$ with pairwise distinct discriminating sets. $\square$

It follows that none of these properties is definable in L_μ. However, it can be shown that unary trace equivalence on acyclic structures is definable in MIC, giving another example of a property separating these two logics. Moreover, it also follows that binary trace equivalence on acyclic structures is not definable in MIC. Since this property is polynomial time decidable and bisimulation invariant, it gives us another instance of a property separating MIC from L_μ^ω. Finally, we note that on arbitrary structures, neither the unary nor the binary trace equivalence problem is definable in MIC. Since the former problem is co-NP-complete and the latter is PSPACE-complete, we do not expect that either is definable in L_μ^ω, but it would be difficult to prove that they are not.

References

1. A. Arnold and D. Niwiński. *Rudiments of μ-calculus*. North Holland, 2001.
2. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
3. A. Dawar, E. Grädel, and S. Kreutzer. Inflationary fixed points in modal logic. In *Proc. of the 10th Conf. on Computer Science Logic (CSL)*, volume 2142 of *LNCS*, pages 277–291. Springer, 2001. Full version at `http://www.cl.cam.ac.uk/~ad260/papers/mic.ps`.
4. F. Gécseg and M. Steinby. Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 1–68. Springer, 1997.
5. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *Proceedings of 7th International Conference on Concurrency Theory CONCUR '96*, volume 1119 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.
6. M. Otto. Bisimulation-invariant Ptime and higher-dimensional mu-calculus. *Theoretical Computer Science*, 224:237–265, 1999.
7. J. Shallit and Y. Breitbart. Automaticity I: Properties of a measure of descriptional complexity. *Journal of Computer and System Sciences*, 53:10–25, 1996.
8. C. Stirling. *Modal and Temporal Properties of Processes*. Springer, 2001.
9. W. Thomas. On logics, tilings, and automata. In J. Leach et al., editor, *Automata, Languages, and Programming*, Lecture Notes in Computer Science Nr. 510, pages 441–453. Springer-Verlag, 1991.
10. W. Thomas. Finite-state recognizability and logic: from words to graphs. In *13th World Computer Congress 94*, volume 1, pages 499–506. Elsevier Science, 1994.
11. W. Thomas. Languages, automata and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 389–455. Springer, 1997.

An Automata-Theoretic Approach to Constraint LTL

Stéphane Demri[1] and Deepak D'Souza[2,⋆]

[1] Lab. Spécification et Vérification, ENS de Cachan & CNRS UMR 8643
61, av. Pdt. Wilson, 94235 Cachan Cedex, France
demri@lsv.ens-cachan.fr
[2] Chennai Mathematical Institute
92 G. N. Chetty Road, Chennai 600 017, India
deepak@cmi.ac.in

Abstract. We consider an extension of linear-time temporal logic (LTL) with constraints interpreted over a concrete domain. We use a new automata-theoretic technique to show PSPACE decidability of the logic for the constraint systems $(\mathbb{Z}, <, =)$ and $(\mathbb{N}, <, =)$. We also give an automata-theoretic proof of a result of Balbiani and Condotta [BC02] for constraint systems that satisfy a "completion" property.

1 Introduction

We consider in this paper an extension of Linear-time Temporal Logic (LTL) [Pnu77] with atomic assertions from a constraint system $\mathcal{D}$. The logic is denoted CLTL($\mathcal{D}$), for Constraint LTL parameterised by the constraint system $\mathcal{D}$, and is obtained from LTL by replacing propositions by atomic constraints in $\mathcal{D}$. In classical LTL variables represent propositions and models are sequences of propositional valuations for these variables. These models can be viewed as having a "spatial" axis (here the elements $\top$ and $\bot$), along which the variables move. In CLTL($\mathcal{D}$), the spatial axis for the models will comprise elements from the domain of $\mathcal{D}$. For example, in CLTL$(\mathbb{N}, <, =)$ one is allowed to use atomic constraints involving $<$ and $=$, and variables ranging over natural numbers. The formula $\Box(x < y)$ in CLTL$(\mathbb{N}, <, =)$ is interpreted over a sequence of $\mathbb{N}$-valuations for the variables x and y, and asserts that at every point in the future, the value of the variable x is less than the value of y. This formula is of course satisfiable, and a candidate satisfying model is $ss\cdots$, where s assigns 1 to x and 2 to y.

Constraint temporal logics have been introduced and studied by logicians in the field of knowledge representation [WZ02]. Spatio-temporal logics, as they are better known there, involve a hybrid of temporal logic and constraint systems, with varying degrees of interaction. For instance, one may be permitted to refer to the value of a variable x in the *next* time instant, leading to constraints of the form $(x > Ox)$. More generously, one may permit $(x > \Diamond y)$, which asserts that the current value of x exceeds the value of *some* future value of y.

⋆ Part of this work was done during this author's visit to LSV, ENS-Cachan.

While research in the area has focused mainly on logics involving constraint systems that have as domains intervals [All83,BC02] and regions [RCC92,WZ00], with a variety of decidability and complexity results, there has been little progress with commonly used constraint systems of the form $(D, <, =)$, with D as the integers $\mathbb{Z}$, or the natural numbers $\mathbb{N}$. Comon and Cortier [CC00] consider a constraint system with the reals $\mathbb{R}$ as the underlying domain and constraints of the form $x < Oy + 2$, which they show is undecidable. They identify a decidable fragment of the logic by restricting the use of the "Until" operator. Balbiani and Condotta [BC02] prove a general decidability result for constraint systems that satisfy a "completion" property – essentially that, given a finite set of consistent constraints S, a valuation for a subset of variables which satisfies the constraints involving only those variables, can be extended to a valuation which satisfies S. The constraint system $(\mathbb{R}, <, =)$ satisfies this property, and the decidability of CLTL$(\mathbb{R}, <, =)$ in PSPACE follows from this result.

In this paper we focus mainly on the constraint systems $(\mathbb{Z},<,=)$ and $(\mathbb{N},<,=)$. As in [BC02], we restrict the interaction between the temporal operators and variables to terms of the form $OO \cdots Ox$. Our approach is automata-theoretic in nature, and we hope that the benefits of such an approach will be evident to the reader. In particular, it facilitates a transparent argument for the result of [BC02] for constraint systems satisfying the completion property. For such constraint systems $\mathcal{D}$ one can further show that the set $L^{\mathcal{D}}(\varphi)$, of models which satisfy a given formula φ, is ω-regular – i.e. it can be described by an automaton on infinite words. This parallels the results for classical LTL [VW86].

The constraint systems $(\mathbb{Z}, <, =)$ and $(\mathbb{N}, <, =)$ are more interesting from a technical point of view. As a typical illustration, the formula $\Box(x > Ox)$ has no (infinite) $\mathbb{N}$-models, while it does have $\mathbb{R}$-models and $\mathbb{Z}$-models. These constraint systems don't satisfy the completion property, and, as we show in Sec. 6, the set of models of a formula in these logics is *not*, in general, ω-regular. Nevertheless, we show that the satisfiability problems for the logics CLTL$(\mathbb{Z}, <, =)$ and CLTL$(\mathbb{N}, <, =)$ are decidable, and in fact PSPACE-complete. Our approach is automata-theoretic, in that we associate with a given formula φ, an automaton $\mathcal{A}^D_\varphi$ which is non-empty iff φ has a D-model (for $D = \mathbb{Z}, \mathbb{N}$). The technique used is as follows: Find an ω-regular superset M of $L^D(\varphi)$, which has the property that all its ultimately periodic words (those of the form $\tau \cdot \delta^\omega$) are also in $L^D(\varphi)$. This property guarantees that M is non-empty iff $L^D(\varphi)$ is. One can then check the emptiness of M using standard automata theory techniques. This lets us decide the D-satisfiability of φ.

The model checking problem for CLTL reduces easily to that of classical LTL. We address this briefly in Sec. 3, and focus on satisfiability in the sequel. Proofs omitted due to lack of space can be found in [DD02].

2 Preliminaries

A *(concrete) constraint system* is of the form $\mathcal{D} = (D, R_1 \ldots, R_n, \mathcal{I})$, where D is a non-empty set called a *domain*, and each R_i is a predicate symbol of arity

a_i, with $\mathcal{I}(R_i) \subseteq D^{a_i}$ being its interpretation. We will suppress the mention of $\mathcal{I}$ whenever it is clear from the context. Let us fix such a constraint system $\mathcal{D}$ for the rest of this section.

An *(atomic) $\mathcal{D}$ constraint* over a finite set of variables U is of the form $R_i(x_1, \ldots, x_{a_i})$ where each $x_i \in U$. A D-valuation over U is a map $s : U \to D$. Let $c = R_i(x_1, \ldots, x_{a_i})$ be a $\mathcal{D}$ constraint over U, and let s be a D-valuation over U. Then we say s satisfies c, written $s \models_{\mathcal{D}} c$, iff $(s(x_1), \ldots, s(x_{a_i})) \in \mathcal{I}(R_i)$.

The temporal logic we consider will use atomic constraints over terms which refer to the values of variables in succeeding time points. Let V be a finite set of variables which we will use in formulas of our logic, and which we fix in the rest of this paper. An *(atomic) term constraint* in $\mathcal{D}$ is of the form

$$R_i(O^{n_1}x_1, \ldots, O^{n_{a_i}}x_{a_i})$$

where $x_1, \ldots, x_{a_i} \in V$, $n_1, \ldots, n_{a_i} \in \mathbb{N}$. Here O^i stands for the juxtaposition of the next-state operator O i times, with O^0x representing just x. By the *O-length* of a term constraint c we will mean the value $i+1$ where i is the largest j for which O^j occurs in c. A term constraint in $\mathcal{D}$ is interpreted over a sequence of D-valuations. A D-valuation sequence is represented as a function $\sigma : \mathbb{N} \to \mathit{val}(D)$, where $\mathit{val}(D)$ denotes the set of D-valuations over V. We say a D-valuation sequence σ satisfies a term constraint $c = R_i(O^{n_1}x_1, \ldots, O^{n_{a_i}}x_{a_i})$, in $\mathcal{D}$, written $\sigma \models_{\mathcal{D}} c$, iff

$$(\sigma(n_1)(x_1), \ldots, \sigma(n_{a_i})(x_{a_i})) \in \mathcal{I}(R_i).$$

For $i, j \in \mathbb{N}$, and $i \leq j$, let us denote by $\sigma^{i,j}$ the finite valuation sequence $\sigma(i), \sigma(i+1), \ldots, \sigma(j)$. The truth of c in σ is thus determined by $\sigma^{0,k-1}$ where k is the O-length of c. A term constraint c can thus be viewed as a constraint over the variables $V \times \{0, \ldots, k-1\}$, where k is the O-length of c.

We now introduce CLTL($\mathcal{D}$), the constraint linear-time temporal logic, parameterised by the constraint system $\mathcal{D}$. The syntax of CLTL($\mathcal{D}$) is given by:

$$\varphi ::= c \mid \neg\varphi \mid (\varphi \vee \varphi) \mid O\varphi \mid (\varphi U \varphi)$$

where c is a term constraint in $\mathcal{D}$.

Models for CLTL($\mathcal{D}$) formulas are D-valuation sequences. Let φ be a CLTL($\mathcal{D}$) formula, and σ a D-valuation sequence. The satisfaction relation $\sigma \models \varphi$ is defined inductively below. We use the notation σ^i to denote the i-th suffix of σ given by $\sigma^i(l) = \sigma(l+i)$ for all $l \in \mathbb{N}$.

$\sigma \models c$ iff $\sigma \models_{\mathcal{D}} c$
$\sigma \models \neg\alpha$ iff $\sigma \not\models \alpha$
$\sigma \models \alpha \vee \beta$ iff $\sigma \models \alpha$ or $\sigma \models \beta$
$\sigma \models O\alpha$ iff $\sigma^1 \models \alpha$
$\sigma \models \alpha U \beta$ iff $\exists k \in \mathbb{N}$ such that $\sigma^k \models \beta$ and $\forall i : 0 \leq i < k,\ \sigma^i \models \alpha$.

The semantics $\models_{\mathrm{LTL}}$ of classical LTL are given in a similar manner using propositions instead of constraints and models as propositional valuation sequences. $\Diamond$ and $\Box$ will stand for the usual temporal abbreviations for "sometimes" and "always", with the semantics $\Diamond\alpha \equiv (\top U \alpha)$, and $\Box\alpha \equiv \neg(\top U \neg\alpha)$.

As an example, consider the constraint system $(\mathbb{N}, <, =)$ with the usual interpretation of the symbols '$<$' and '$=$'. Let V be the set of variables $\{x, y\}$ and let us represent a valuation s over V as $\langle s(x), s(y)\rangle$. Then the formula $\Box(x < Oy)$ is satisfied by the $\mathbb{N}$-valuation sequence $\sigma = \langle 1, 2\rangle\langle 2, 4\rangle\langle 3, 6\rangle \cdots$, but not by $\sigma' = \langle 1, 1\rangle\langle 2, 2\rangle\langle 2, 2\rangle \cdots$.

The semantics of CLTL($\mathcal{D}$) can also be given in the sense of Prop. 1 below, in terms of what we call *frame sequences*. A *frame* over a set of variables U, w.r.t. a constraint system $\mathcal{D}$, is essentially a maximally consistent set of $\mathcal{D}$ constraints over U. More precisely, let $\mathit{frame}_{\mathcal{D}}(s, U)$ denote the set of $\mathcal{D}$ constraints over U satisfied by a D-valuation s over U. Then a set S of $\mathcal{D}$ constraints, is a *frame* over the variables U, w.r.t. $\mathcal{D}$, if there exists a D-valuation s over U such that $S = \mathit{frame}_{\mathcal{D}}(s, U)$. The *frame checking problem* for a constraint system $\mathcal{D}$ is to decide whether a given set of $\mathcal{D}$ constraints S, over a finite set of variables U, is a frame over U, w.r.t. $\mathcal{D}$.

In the context of terms in the logic it is convenient to define the notion of a *k-frame* which is essentially a frame over the set of variables $V \times \{0, \ldots, k-1\}$. Let $P_k^{\mathcal{D}}$ denote the set of propositions p_c such that c is a term constraint in $\mathcal{D}$ of O-length at most k. Let σ be a D-valuation sequence. We use $k\text{-}\mathit{frame}_{\mathcal{D}}(\sigma)$ to denote the set of all propositions $p_c \in P_k^{\mathcal{D}}$ such that $\sigma \models_{\mathcal{D}} c$. A *$k$-frame* w.r.t. $\mathcal{D}$ is a propositional valuation v over the set of propositions $P_k^{\mathcal{D}}$ (represented as a subset of $P_k^{\mathcal{D}}$) such that there exists a D-valuation sequence σ with $v = k\text{-}\mathit{frame}_{\mathcal{D}}(\sigma)$.

We say a pair of k-frames (v, v') is *locally consistent* if for all R_i, and for all $n_1, \ldots, n_{a_i} \in \{1, \ldots, k-1\}$:

$$R_i(O^{n_1}x_1, \ldots, O^{n_{a_i}}x_{a_i}) \in v \text{ iff } R_i(O^{n_1-1}x_1, \ldots, O^{n_{a_i}-1}x_{a_i}) \in v'.$$

(For readability, we write c instead of p_c.) Accordingly, a k-frame sequence ρ is locally consistent if for all $l \in \mathbb{N}$, the pairs $(\rho(l), \rho(l+1))$ are locally consistent. A given D-valuation sequence σ induces, for a given k, a canonical locally consistent k-frame sequence ρ, w.r.t. $\mathcal{D}$, denoted $k\text{-}\mathit{fs}_{\mathcal{D}}(\sigma)$, and given by $\rho(i) = k\text{-}\mathit{frame}_{\mathcal{D}}(\sigma^i)$. We extend k-*fs* to act on sets of valuation sequences in the natural way. A k-frame sequence ρ will be said to *admit* a D-valuation sequence, if there exists a D-valuation sequence σ such that $k\text{-}\mathit{fs}_{\mathcal{D}}(\sigma) = \rho$.

We say a CLTL($\mathcal{D}$) formula φ has O-length k if the largest O-length of constraints in φ is k. Thus the O-length of the formula $\Box(x < Oy)$ is 2. For a CLTL($\mathcal{D}$) formula φ, let $\widehat{\varphi}$ denote the classical LTL formula obtained from φ by replacing each constraint c by p_c. Then the following proposition is a direct consequence of the semantics of CLTL($\mathcal{D}$) and classical LTL.

Proposition 1. *Let φ be a* CLTL($\mathcal{D}$) *formula of O-length k. Let σ be a D-valuation sequence, and let $\rho = k\text{-}\mathit{fs}_{\mathcal{D}}(\sigma)$. Then $\sigma \models \varphi$ iff $\rho \models_{\mathrm{LTL}} \widehat{\varphi}$.* $\Box$

By the above proposition, a CLTL($\mathcal{D}$) formula φ of O-length k cannot differentiate between D-valuations that induce the same k-frame sequences. We define $L_{\mathit{kfs}}^{\mathcal{D}}(\varphi) = \{k\text{-}\mathit{fs}_{\mathcal{D}}(\sigma) \mid \sigma \models \varphi\}$. Stated differently, $L_{\mathit{kfs}}^{\mathcal{D}}(\varphi)$ is the set of all k-frame sequences (w.r.t. $\mathcal{D}$) that satisfy $\widehat{\varphi}$ and admit a D-valuation.

3 Model Checking

Let $\mathcal{D} = (D, R_1, \ldots, R_n, \mathcal{I})$ be a given constraint system. A *D-Kripke structure* is of the form $(Q, \longrightarrow, l)$ where $(Q, \longrightarrow)$ is a finite directed graph, and $l : Q \longrightarrow val(D)$ is a labelling of states with D-valuations. For a D-Kripke structure $M = (Q, \longrightarrow, l)$ and a state $q \in Q$, we define $L(M, q)$ to be the set of all D-valuation sequences which appear as labels along an infinite path in M beginning at q.

Let $\varphi \in \text{CLTL}(\mathcal{D})$. We say $M, q \models \varphi$ iff $\sigma \models \varphi$ for each $\sigma \in L(M, q)$. The model checking problem for CLTL($\mathcal{D}$) is: given a program modelled as a D-Kripke structure M, a state q in M, and a formula φ in CLTL($\mathcal{D}$); does $M, q \models \varphi$?

In the instance of the model checking problem above, let φ have O-length k. Then, by Prop. 1, $M, q \models \varphi$ iff $\rho \models_{\text{LTL}} \widehat{\varphi}$ for every k-frame sequence ρ induced by a D-valuation sequence $\sigma \in L(M, q)$. We can define a classical $P_k^{\mathcal{D}}$-labelled Kripke structure $M_k = (T, \longrightarrow, m)$, which (for suitable choices of initial states) generates precisely the language $k\text{-}fs_{\mathcal{D}}(L(M, q))$. Define T to be the set of k-length paths in M. We define $q_0 \cdots q_{k-1} \longrightarrow r_0 \cdots r_{k-1}$ iff for each $i \in \{1, \ldots, k-1\}$, $q_i = r_{i-1}$. Define $m(q_0 \cdots q_{k-1}) = k\text{-}frame_{\mathcal{D}}(l(q_0) \cdots l(q_{k-1}))$.

The model checking problem for CLTL($\mathcal{D}$) now reduces to the model checking problem for LTL in the following sense: $(M, q) \models \varphi$ iff $(M_k, t) \models_{\text{LTL}} \widehat{\varphi}$ for each $t \in T$ of the form $q_0 \cdots q_{k-1}$ with $q_0 = q$. This gives us a way to decide, in PSPACE, the model checking problem for constraint systems that are "well-behaved" in that the satisfiability problem for the predicates R_i are decidable in PSPACE.

Proposition 2. *For every well-behaved constraint system $\mathcal{D}$, the model checking problem for* CLTL($\mathcal{D}$) *is in* PSPACE. $\square$

4 A General Satisfiability Result

For a constraint system $\mathcal{D}$, the satisfiability problem for CLTL($\mathcal{D}$) is: given a CLTL($\mathcal{D}$) formula φ, does there exist a D-valuation sequence which satisfies φ? Our approach to answering this problem is to decide whether there exists a k-frame sequence ρ (where k is the O-length of φ), which satisfies $\widehat{\varphi}$ *and* admits a D-valuation sequence. Since such a k-frame sequence must be locally consistent, we can restrict our attention to k-frame sequences which are locally consistent.

In general, a locally consistent k-frame sequence may not admit a D-valuation sequence (see for instance the constraint system $(\mathbb{Z}, <, =)$ in Sec. 6). However, when the constraint system satisfies the "completion" property described below, locally consistent k-frame sequences do admit D-valuation sequences.

A constraint system $\mathcal{D}$ is said to satisfy the *completion* property if given

- a frame S over a finite set of variables U w.r.t. $\mathcal{D}$,
- a subset U' of U, and
- a valuation s' over U', such that $S \restriction U' = frame_{\mathcal{D}}(s', U')$ (by $S \restriction U'$ we mean the subset of constraints in S which use only variables in U');

then there exists a valuation s over U which extends s' and satisfies $S = \mathit{frame}_{\mathcal{D}}(s, U)$. The constraint systems $(D, <, =)$ for $D = \mathbb{Q}, \mathbb{R}$ are examples of systems that satisfy the completion property.

Proposition 3. *Let $\mathcal{D}$ be a constraint system which satisfies completion. Then every locally consistent k-frame sequence w.r.t. $\mathcal{D}$ admits a D-valuation sequence.*

Proof. Let $\rho = v_0 v_1 \ldots$ be a locally consistent k-frame sequence w.r.t. $\mathcal{D}$. We define a D-valuation sequence $\sigma = s_0 s_1 \cdots$, with the property that $\rho = k\text{-}\mathit{fs}_{\mathcal{D}}(\sigma)$, as follows. We view each v_i as a set of constraints over the variables $U_i = V \times \{i, \ldots, i+k-1\}$. Since each v_i is a k-frame, for each i there exists a D-valuation t_i over U_i such that $v_i = \mathit{frame}_{\mathcal{D}}(t_i, U_i)$. Define $s_0(x) = t_0(x, 0), \ldots, s_{k-1}(x) = t_0(x, k-1)$, for all $x \in V$. Now consider the set of constraints v_1. We know that v_1 is a frame over U_1. Further consider the subset $U' = V \times \{1, \ldots, k-1\}$ of U_1. Clearly the restriction t' of the valuation t_0 to U', is such that $v_1 \upharpoonright U' = \mathit{frame}_{\mathcal{D}}(t', U')$. Thus, by the completion property we have a D-valuation t'' which extends t' to the variables $V \times \{k\}$, and satisfies $v_1 = \mathit{frame}_{\mathcal{D}}(t'', U_1)$. We can now define $s_k(x) = t''(x, k)$ for each $x \in V$. This argument can be extended repeatedly to define the rest of σ. Clearly, σ has the property that $k\text{-}\mathit{fs}_{\mathcal{D}}(\sigma) = \rho$, and hence ρ admits the D-valuation sequence σ. □

We can now state an interesting result regarding the language of k-frames defined by a CLTL($\mathcal{D}$) formula φ. Recall that a Büchi automaton over an alphabet A is simply a classical automaton $\mathcal{A} = (Q, q_0, \longrightarrow, F)$ over A, but with the set of final states F used as a acceptance condition on infinite words $\alpha \in A^\omega$. A run $\rho : \mathbb{N} \to Q$ on α is accepting if $\rho(i) \in F$ for infinitely many $i \in \mathbb{N}$. $L(\mathcal{A})$ is the set of infinite words accepted by $\mathcal{A}$, and $L \subseteq A^\omega$ is termed ω-regular if $L = L(\mathcal{A})$ for some Büchi automaton $\mathcal{A}$ over A (see [Tho90]).

Proposition 4. *Let $\mathcal{D}$ be a constraint system satisfying the completion property. Let φ be a* CLTL($\mathcal{D}$) *formula. Then $L^{\mathcal{D}}_{\mathit{kfs}}(\varphi)$ is ω-regular.*

Proof. Let φ be of O-length k. We define a Büchi automaton $\mathcal{A}^{\mathcal{D}}_{\varphi}$ over the alphabet $2^{(P^{\mathcal{D}}_k)}$, such that $L(\mathcal{A}^{\mathcal{D}}_{\varphi}) = L^{\mathcal{D}}_{\mathit{kfs}}(\varphi)$. Define $\mathcal{A}^{\mathcal{D}}_{\varphi} = \mathcal{A}_{\widehat{\varphi}} \cap \mathcal{A}^{\mathcal{D},k}_{lc}$, where $\mathcal{A}_{\widehat{\varphi}}$ is the Vardi-Wolper automaton [VW86] for the LTL formula $\widehat{\varphi}$, and $\mathcal{A}^{\mathcal{D},k}_{lc}$ is a Büchi automaton over $2^{(P^{\mathcal{D}}_k)}$ which accepts locally consistent k-frame sequences. We can define $\mathcal{A}^{\mathcal{D},k}_{lc} = (Q, q_0, \longrightarrow, F)$ where Q is the set of k-frames w.r.t. $\mathcal{D}$, along with a separate start state q_0; $\longrightarrow$ is given by $q_0 \xrightarrow{v} v$, and $v \xrightarrow{v'} v'$ iff (v, v') is locally consistent (here v and v' range over k-frames w.r.t $\mathcal{D}$); and $F = Q$. □

Prop. 4 gives us a way to decide the satisfiability problem for a constraint system $\mathcal{D}$ which satisfies the completion property, and has a decidable frame checking problem. We can do this by constructing $\mathcal{A}^{\mathcal{D}}_{\varphi}$ and checking for language emptiness. In particular, if $\mathcal{D}$ is such that frame checking can be done in PSPACE, then the satisfiability problem for CLTL($\mathcal{D}$) can be solved in PSPACE. This is because both $\mathcal{A}_{\widehat{\varphi}}$ and $\mathcal{A}^{\mathcal{D},k}_{lc}$ are implicitly defined graphs whose transition relations can be checked (with the above assumption) in PSPACE in the length of φ.

Since the number of states in both $\mathcal{A}_{\widehat{\varphi}}$ and $\mathcal{A}_{lc}^{\mathcal{D},k}$ are exponential in length of φ, reachability of states in $\mathcal{A}_{\varphi}^{\mathcal{D}}$ can be done non-deterministically in PSPACE. Thus, emptiness checking, which essentially involves a couple of reachability checks, can be done non-deterministically in PSPACE in length of φ. This is essentially the result of [BC02].

Theorem 1. *The satisfiability problem for* CLTL($\mathcal{D}$) *when $\mathcal{D}$ satisfies completion and allows frame-checking in* PSPACE*, is in* PSPACE. □

5 Constraint Systems of the Form $(D, <, =)$

In this section we develop some notions which are useful in dealing with constraints systems of the form $(D, <, =)$. The case $D = \mathbb{R}$ is dealt with at the end of the section (the rationals $\mathbb{Q}$ behave very similarly as far as our logic is concerned), while $\mathbb{Z}$ and $\mathbb{N}$ are completed in the next section.

For a constraint system of the form $(D, <, =)$ it is convenient to visualise a frame as a labelled, directed graph. We represent a frame v over a finite set of variables U by a $\{<, =\}$-labelled, directed graph over the vertices U, where we place a '$\sim$'-labelled edge (for $\sim \in \{<, =\}$) from x to y precisely when $x \sim y \in v$. Such a graph clearly satisfies the conditions that:

1. there is an edge between every pair of vertices;
2. if there is '$=$'-labelled edge from x to y then there is also one from y to x;
3. there are no *strict* cycles (i.e. directed cycles with a '$<$'-labelled edge).

Conversely, given an $\{<, =\}$-labelled graph G over a finite set of vertices U which satisfies the above conditions, one can verify that the set of constraints $v_G = \{x \sim y \mid x \xrightarrow{\sim} y \text{ in } G\}$ constitutes a frame over U. In the sequel we will often make use of this graphical representation of frames. Fig. 1 shows the first two 3-frames of a locally consistent 3-frame sequence over variables x, y.

A locally consistent k-frame sequence ρ can be represented as a single $\{<, =\}$-labelled, directed graph G_ρ, which is essentially the super-imposition of the overlapping k-frames (see Fig. 1). More precisely, G_ρ has $V \times \mathbb{N}$ as its set of vertices, and an edge $(x, i) \xrightarrow{\sim} (y, j)$ iff either $i \leq j$ and $(x \sim O^{j-i}y) \in \rho(i)$, or, $i > j$ and $(O^{i-j}x \sim y) \in \rho(j)$. An important property of G_ρ is that it does not contain any strict cycles (see below).

The frame checking problem for $(D, <, =)$ is essentially the cycle detection problem in a directed graph, which is in NLOGSPACE. It is not difficult to argue that $(\mathbb{R}, <, =)$ satisfies the completion property [DD02]. Thus, from Theorem 1 we conclude that the satisfiability problem for CLTL($\mathbb{R}, <, =$) is PSPACE-complete. As a corollary we can also conclude that for a locally consistent k-frame sequence ρ, G_ρ *always* admits an $\mathbb{R}$-valuation and hence contains no strict cycles.

6 Satisfiability for CLTL($\mathbb{Z}, <, =$)

The constraint system $(\mathbb{Z}, <, =)$ does not satisfy the completion property and hence we cannot appeal to Theorem 1 to solve the satisfiability problem for the

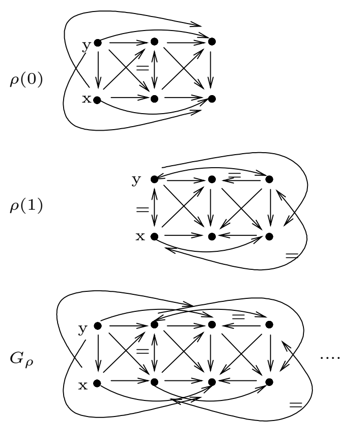


Fig. 1. A 3-frame sequence over the variables $\{x, y\}$.

logic CLTL($\mathbb{Z}, <, =$). In fact, unlike $\mathbb{R}$, the language $L^{\mathbb{Z}}_{kfs}(\varphi)$ is *not*, in general, ω-regular (see Cor. 1). However, we can still solve the satisfiability problem for $\mathbb{Z}$ automata-theoretically. The idea is to define a Büchi automaton $\mathcal{A}^{\mathbb{Z}}_{\varphi}$ which accepts a superset of $L^{\mathbb{Z}}_{kfs}(\varphi)$, with the property that all *ultimately periodic* words in it are also in $L^{\mathbb{Z}}_{kfs}(\varphi)$. It then follows that $L(\mathcal{A}^{\mathbb{Z}}_{\varphi})$ is non-empty iff φ is $\mathbb{Z}$-satisfiable.

We begin with a characterisation of locally consistent k-frames which admit a $\mathbb{Z}$-valuation. Let ρ be a locally consistent k-frame sequence. For a directed path p in G_ρ, let $slen(p)$ (the *strict* length of p) denote the number of '$<$'-labelled edges in p if this number is finite, and ω otherwise. For any two vertices u, v in G_ρ, define $slen(u, v)$ to be the supremum of $slen(p)$ over directed paths p from u to v, if it exists, and ω otherwise.

Lemma 1. *Let ρ be a locally consistent k-frame sequence. Then ρ admits a $\mathbb{Z}$-valuation iff for all $u, v \in G_\rho$, $slen(u, v) < \omega$.*

Proof. If ρ admits a $\mathbb{Z}$-valuation, then clearly $slen(u, v) \leq\mid f(v) - f(u) \mid$, where f is a $\mathbb{Z}$-valuation admitted by ρ. Thus, there cannot exist vertices u and v with $slen(u, v) = \omega$. Conversely, suppose G_ρ satisfies the given condition. Then one can verify that the procedure given below produces a $\mathbb{Z}$-labelling f of the vertices of G_ρ that respects the labels on edges of G_ρ. This in turn implies that ρ admits a $\mathbb{Z}$-valuation. We assume an ordering $\prec$ on variables, and use it to define an ordering of vertices in G_ρ given by $(x, i) \prec (y, j)$ iff $i < j$, or $i = j$ and $x \prec y$.

1. Label the vertices in order. Begin by labelling the first, say $(x, 0)$, by 0.
2. In general, if X is the portion of the graph already labelled, and u is the next vertex to be labelled: if there is a directed path from u to a vertex in X, set $f(u) = \min\{f(v) - slen(u, v) \mid v \in X, \ \exists \text{ a path from } u \text{ to } v\}$, else, set $f(u) = \max\{f(v) + slen(v, u) \mid v \in X, \ \exists \text{ a path from } v \text{ to } u\}$. □

Let ρ be a locally consistent k-frame sequence. An infinite *forward* increasing (respectively decreasing) chain in G_ρ is a sequence $d : \mathbb{N} \to V \times \mathbb{N}$ satisfying:

1. for all $i \in \mathbb{N}$, there is an edge from $d(i)$ to $d(i+1)$ (respectively, an edge from $d(i+1)$ to $d(i)$),
2. for all $i \in \mathbb{N}$, if $d(i)$ is in level j, then $d(i+1)$ is in a level greater than or equal to $j+1$. By the "level" of a vertex (x, i) we mean i.

Such a chain d is *strict* if there exist infinitely many i for which there is a '$<$'-labelled edge from $d(i)$ to $d(i+1)$ (respectively, from $d(i+1)$ to $d(i)$).

Consider now the condition $(C_\mathbb{Z})$ below on a locally consistent k-frame sequence ρ: there *do not* exist vertices u and v in the same k-frame in G_ρ satisfying:

1. there is an infinite forward increasing chain d from u,
2. there is an infinite forward decreasing chain e from v,
3. either d or e is strict, and
4. for each $i, j \in \mathbb{N}$, whenever $d(i)$ and $e(j)$ belong to the same k-frame there is an edge labelled '$<$' from $d(i)$ to $e(j)$.

It follows from Lemma 1 that condition $(C_\mathbb{Z})$ is necessary for ρ to admit a $\mathbb{Z}$-valuation sequence. But it is not sufficient, as witnessed by the 2-frame sequence $\rho^\mathbb{Z}_{bad}$ in Fig. 2. We have shown only the relevant edges in the figure. $\rho^\mathbb{Z}_{bad}$ clearly satisfies condition $(C_\mathbb{Z})$. However, it cannot admit a $\mathbb{Z}$-valuation since there are paths of unbounded strict length from $(x, 0)$ to $(z, 0)$.

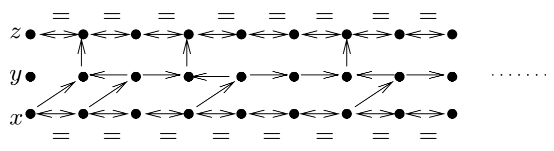


Fig. 2. Frame sequence $\rho^\mathbb{Z}_{bad}$ satisfies $(C_\mathbb{Z})$ but does not admit a $\mathbb{Z}$-valuation.

However, when ρ is an *ultimately periodic* word (i.e. of the form $\tau \cdot \delta^\omega$), this condition is indeed sufficient.

Lemma 2. *Let ρ be an ultimately periodic, locally consistent, k-frame sequence. Then ρ admits a $\mathbb{Z}$-valuation sequence iff ρ satisfies $(C_\mathbb{Z})$.*

Proof. If ρ admits a $\mathbb{Z}$-valuation, then by Lemma 1 ρ must satisfy the condition $(C_\mathbb{Z})$. Conversely, suppose ρ does not admit a $\mathbb{Z}$-valuation. We will show that ρ fails to meet condition $(C_\mathbb{Z})$. By Lemma 1 we know that there must exist two vertices u and v in G_ρ with $slen(u, v) = \omega$.

Now for any $i \in \mathbb{N}$, we can find a path p_i from u to v in G_ρ, of the form $u \xrightarrow{d_i} w \xrightarrow{e_i} v$ where d_i is a forward increasing chain from u to w, e_i is a forward decreasing from v to w, both d_i and e_i have length at least i, and either d_i or e_i has strict length at least i. This must be true, given the bounded width of the graph, and the unbounded strict length of paths from u to v.

We can also assume that u and v are in the same k-frame. Otherwise (say v was ahead of u), using an argument similar to König's Lemma, we can find a vertex u' in the same k-frame as v, with $slen(u', v) = \omega$.

Now let $\rho = \tau \cdot \delta^\omega$. Let there be N nodes in δ. Consider p_i for some $i > |\tau| + N$. We assume here that d_i is the one with at least i '$<$' edges (a symmetric argument holds when e_i has this property). Now it must be the case that d_i visits a vertex x in a δ block twice, *with at least one* '$<$' edge between the two occurrences (see Fig. 3). Let this segment of d_i, which begins at x and ends at x (in different δ blocks) be q. Note that the "future" of the vertices x in different δ blocks are exactly the same by the periodic nature of ρ. Now consider the path f from u given by the initial part of d_i to x, then q followed by q, ad infinitum. This is a strict infinite forward increasing chain from u in G_ρ. In a similar manner, the path e_i gives rise to an infinite forward (not necessarily strict) decreasing chain g from v, via an ultimately periodic concatenation of a path segment r between occurrences of y.

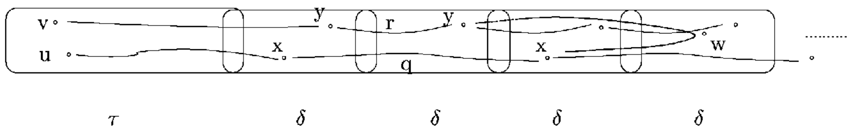


Fig. 3. Infinite paths extracted from p_i.

Finally we must argue that for each vertex u' in f and v' in g, whenever they are in the same k-frame, there is an edge from u' to v' labelled $<$. The interesting case is when u' and v' lie on some copies of the path segments q and r (see Fig. 4). The initial occurrences of the segments q and r in f and g were part of the path p_i. Hence there is a path t from x to w to y. Further, this path is strict, since by the choice of i, the path from x to w must contain at least one '$<$'. Now, there must be a point in the future (a multiple of the lcm of the lengths of q and r) which occurs after u' and v', and where the path t is duplicated. Thus we have a path from u' to x, followed by a strict path t, followed by a path from y to v' in G_ρ. Since there must be an edge between u' and v' (being in the same k-frame), the edge must be strict and directed from u' to v' (otherwise G_ρ would have a strict cycle!). □

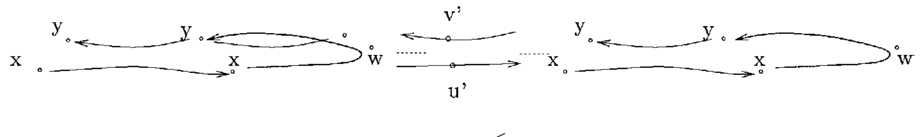


Fig. 4. $u' \xrightarrow{<} v'$.

Let us now define $\mathcal{A}^{\mathbb{Z}}_{\varphi} = \mathcal{A}_{\widehat{\varphi}} \cap \mathcal{A}^{\mathbb{Z},k}_{lc} \cap \mathcal{A}^{\mathbb{Z}}$, where $\mathcal{A}_{\widehat{\varphi}}$ and $\mathcal{A}^{\mathbb{Z},k}_{lc}$ are as in Sec. 4, and $\mathcal{A}^{\mathbb{Z}}$, described below, accepts k-frame sequences which satisfy $(C_{\mathbb{Z}})$. Let $\mathcal{B}$ be a Büchi automaton over the alphabet of k-frames, which simply checks the negation of condition ($C_{\mathbb{Z}}$). Thus $\mathcal{B}$ non-deterministically guesses the vertices u and v, and then verifies the conditions (1)–(4). We non-deterministically choose in the beginning, whether to signal (via the Büchi condition) each time the d or e path sees an edge labelled '<'. The automaton $\mathcal{A}^{\mathbb{Z}}$ is now just the complement of $\mathcal{B}$.

Corollary 1. *There exist φ in* CLTL($\mathbb{Z}, <, =$) *for which $L^{\mathbb{Z}}_{kfs}(\varphi)$ is not ω-regular.*

Proof. Consider the formula $\varphi = \Diamond((x < Oy) \wedge (y < z))$. The frame sequence $\rho^{\mathbb{Z}}_{bad}$ of Fig. 2 is a model for the LTL formula $\widehat{\varphi}$. Suppose $L^{\mathbb{Z}}_{kfs}(\varphi)$ were ω-regular. Then, $L = ((2^{P^{\mathcal{D}}_k})^{\omega} - L^{\mathbb{Z}}_{kfs}(\varphi)) \cap L(\mathcal{A}^{\mathbb{Z}}_{\varphi})$ is also ω-regular. Now, $\rho^{\mathbb{Z}}_{bad} \notin L^{\mathbb{Z}}_{kfs}(\varphi)$ and $\rho^{\mathbb{Z}}_{bad} \in L(\mathcal{A}^{\mathbb{Z}}_{\varphi})$. Hence $\rho^{\mathbb{Z}}_{bad} \in L$. But L is ω-regular, and hence there must exist an ultimately periodic word ρ in L. But $\rho \in L(\mathcal{A}^{\mathbb{Z}}_{\varphi})$ and hence satisfies ($C_{\mathbb{Z}}$). By Lemma 2, ρ admits a $\mathbb{Z}$-valuation sequence. Since $\rho \in L(\mathcal{A}_{\widehat{\varphi}})$, we can conclude that $\rho \in L^{\mathbb{Z}}_{kfs}(\varphi)$. This is a contradiction. $\square$

Lemma 3. *A* CLTL($\mathbb{Z}$) *formula φ is $\mathbb{Z}$-satisfiable iff $L(\mathcal{A}^{\mathbb{Z}}_{\varphi})$ is non-empty.*

Proof. Suppose φ is $\mathbb{Z}$-satisfiable. Let $\sigma \models \varphi$, and let $\rho = k\text{-}fs(\sigma)$, where k is the O-length of φ. By Prop. 1, $\rho \in L(\mathcal{A}_{\widehat{\varphi}})$. We know that $\rho \in L(\mathcal{A}^{\mathbb{Z},k}_{lc})$. Further, by Lemma 1, G_ρ satisfies ($C_{\mathbb{Z}}$) and hence $\rho \in L(\mathcal{A}^{\mathbb{Z}})$. Thus $\rho \in L(\mathcal{A}_{\widehat{\varphi}}) \cap L(\mathcal{A}^{\mathbb{Z},k}_{lc}) \cap L(\mathcal{A}^{\mathbb{Z}})$. Hence $\rho \in L(\mathcal{A}^{\mathbb{Z}}_{\varphi})$.

Conversely, suppose $\mathcal{A}^{\mathbb{Z}}_{\varphi}$ accepts a word ρ. Then it must accept an ultimately periodic word ρ' (by nature of the acceptance condition). Now ρ' is locally consistent and satisfies ($C_{\mathbb{Z}}$). Hence by Lemma 2, ρ' must admit a $\mathbb{Z}$-valuation, say σ. Since $\rho' \models_{\mathrm{LTL}} \widehat{\varphi}$, by Prop. 1, $\sigma \models \varphi$. Thus φ is $\mathbb{Z}$-satisfiable. $\square$

Theorem 2. *The satisfiability problem for* CLTL($\mathbb{Z}, <, =$) *is* PSPACE*-complete.*

Proof. The number of states in $\mathcal{B}$ above can be seen to be polynomial in $|\varphi|$. Hence $\mathcal{A}^{\mathbb{Z}}$, which is the complement of $\mathcal{B}$, is at most exponential in $|\varphi|$ using, say, the construction of Safra [Saf88]. The number of states in both $\mathcal{A}_{\widehat{\varphi}}$ and $\mathcal{A}^{\mathbb{Z},k}_{lc}$ is also exponential in $|\varphi|$. Further, the transition relation of each of these components is implicitly defined in terms of φ and can be computed in polynomial space in $|\varphi|$. It follows that the emptiness of $\mathcal{A}^{\mathbb{Z}}_{\varphi}$ can be decided in PSPACE in the length of φ. PSPACE-hardness follows from that of LTL [SC85]. $\square$

We close this section with an outline of the proof that the satisfiability problem for CLTL($\mathbb{N}, <, =$) is PSPACE-complete. The treatment of this case is very similar to that of $\mathbb{Z}$. We outline the counterparts of the lemmas for the $\mathbb{Z}$ case. Define, for a vertex u in G_ρ, $sdlen(u)$ to be the supremum of $slen(p)$ over directed paths p from some vertex v to u in G_ρ. Then the counterpart of Lemma 1 is that a locally consistent k-frame sequence ρ admits an $\mathbb{N}$-valuation iff for all $u \in G_\rho, sdlen(u) < \omega$.

Let ($C_{\mathbb{N}}$) denote the following condition on a locally consistent k-frame sequence ρ: G_ρ satisfies ($C_{\mathbb{Z}}$) *and* it does not contain a strict infinite forward descending chain. One can now prove, using an argument similar to the proof of Lemma 2, that: an ultimately periodic, locally consistent, k-frame sequence ρ admits an $\mathbb{N}$-valuation sequence iff ρ satisfies $(C_{\mathbb{N}})$. Now define an automaton $\mathcal{A}_{\varphi}^{\mathbb{N}}$ as in the $\mathbb{Z}$ case, so that the formula φ is $\mathbb{N}$-satisfiable iff $L(\mathcal{A}_{\varphi}^{\mathbb{N}})$ is non-empty.

Theorem 3. *The satisfiability problem for* CLTL$(\mathbb{N}, <, =)$ *is* PSPACE*-complete.*

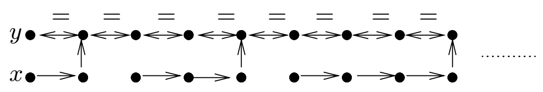


Fig. 5. Frame sequence $\rho_{bad}^{\mathbb{N}}$ satisfies $(C_{\mathbb{N}})$ but does not admit an $\mathbb{N}$-valuation.

Finally, the frame sequence $\rho_{bad}^{\mathbb{N}}$ of Fig. 5 allows us to argue that there is a CLTL$(\mathbb{N}, <, =)$ formula φ for which $L_{kfs}^{\mathbb{N}}(\varphi)$ is not ω-regular.

References

All83. J. F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–846, 1983.

BC02. Ph. Balbiani and J.F. Condotta. Computational complexity of propositional linear temporal logics based on qualitative spatial or temporal reasoning. In *Frontiers of Combining Systems (FroCoS'02)*, pages 162–176, 2002.

CC00. H. Comon and V. Cortier. Flatness is not a weakness. In *14 Int. Workshop Computer Science Logic*, pages 262–276. LNCS, vol. 1862. Springer, 2000.

DD02. S. Demri and D. D'Souza. An automata-theoretic approach to constraint LTL. *CMI Internal Report TCS-02-01*, 2002. `http://www.cmi.ac.in`.

Pnu77. A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on Foundation of Computer Science*, pages 46–57, 1977.

RCC92. D. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In *3rd Conference on Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufman, 1992.

Saf88. S. Safra. On the complexity of ω-automata. *Proc. 29th IEEE Symp. on Foundations of Computer Science*, pages 319–327, 1988.

SC85. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *JACM*, 32(3):733–749, 1985.

Tho90. W. Thomas. Automata on infinite objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B. Elsevier, 1990.

VW86. M. Vardi and P. Wolper. An automata theoretic approach to automatic program verification. In *Logic in Computer Science*, pages 332–334. IEEE, 1986.

WZ00. F. Wolter and M. Zakharyaschev. Spatio-temporal representation and reasoning based on RCC-8. In *KR'00*, pages 3–14. Morgan Kaufmann, 2000.

WZ02. F. Wolter and M. Zakharyaschev. Qualitative spatio-temporal representation and reasoning: a computational perspective. In *Exploring Artificial Intelligence in the New Millenium*, 2002. To appear.

Hardness Results for Multicast Cost Sharing
(Extended Abstract)*

Joan Feigenbaum[1,**], Arvind Krishnamurthy[1,***],
Rahul Sami[1,†], and Scott Shenker[2,‡]

[1] Yale University, Computer Science Dept., New Haven, CT 06520-8285.
{feigenbaum,arvind,sami}@cs.yale.edu
[2] ICSI, 1947 Center Street, Berkeley, CA 94704-1198
shenker@icsi.berkeley.edu

Abstract. We continue the study of *multicast cost sharing* from the viewpoints of both computational complexity and economic mechanism design. We provide fundamental lower bounds on the network complexity of group-strategyproof, budget-balanced mechanisms. We also extend a classical impossibility result in game theory to show that no strategyproof mechanism can be both approximately efficient and approximately budget-balanced.

1 Introduction

In the standard *unicast* model of Internet transmission, each packet is sent to a single destination. Although unicast service has great utility and widespread applicability, it cannot efficiently transmit popular content, such as movies or concerts, to a large number of receivers; the source would have to transmit a separate copy of the content to each receiver independently. The *multicast* model of Internet transmission relieves this problem by setting up a shared delivery tree spanning all the receivers; packets sent down this tree are replicated at branch points so that no more than one copy of each packet traverses each link. Multicast thus greatly reduces the transmission costs involved in reaching large user populations.

The large-scale, high-bandwidth multicast transmissions required for movies and other potential sources of revenue are likely to incur substantial transmission costs. The costs when using the unicast transmission model are separable in that the total cost of the transmission is merely the sum of the costs of transmission to

* This work was supported by the DoD University Research Initiative (URI) program administered by the Office of Naval Research under Grant N00014-01-1-0795. A full version of this paper has been submitted for journal publication [4].

** Supported in part by ONR grants N00014-01-1-0795 and N00014-01-1-0447 and NSF grants CCR-0105337, CCR-TC-0208972, ANI-0207399, and ITR-0219018.

*** Supported in part by NSF grants CCR-9985304, ANI-0207399, and CCR-0209122.

† Supported by NSF Career grant CCR-9702980 and ONR grant N00014-01-1-0795.

‡ Supported in part by NSF grants ANI-9730162, ITR-0081698, ITR-0121555, and ANI-0207399.

each receiver. Multicast's use of a shared delivery tree greatly reduces the overall transmission costs, but, because the total cost is now a submodular but nonlinear function of the set of receivers, it is not clear how to share the costs among the receivers. A recent series of papers has addressed the problem of cost sharing for Internet multicast transmissions. In the first paper on the topic, Herzog, Shenker, and Estrin [9], considered axiomatic and implementation aspects of the problem. Subsequently, Moulin and Shenker [14] studied the problem from a purely economic point of view. Several more recent papers [5,2,1,7] adopt the *distributed algorithmic mechanism design* approach, which augments a game-theoretic perspective with distributed computational concerns[1]. In this paper, we extend the results of [5] by considering a more general computational model and approximate solutions. We also extend a classic impossibility [8] result by showing that no strategyproof mechanism can be both approximately efficient and approximately budget-balanced.

1.1 Multicast Cost Sharing Model

We use the multicast-transmission model of [5]: There is a user population P residing at a set of network nodes N, which are connected by bidirectional network links L. The multicast flow emanates from a source node $\alpha_s \in N$; given any set of receivers $R \subseteq P$, the transmission flows through a *multicast tree* $T(R) \subseteq L$ rooted at α_s and spans the nodes at which users in R reside. It is assumed that there is a *universal* tree $T(P)$ and that, for each subset $R \subseteq P$, the multicast tree $T(R)$ is merely the minimal subtree of $T(P)$ required to reach the elements in R. This approach is consistent with the design philosophy embedded in essentially all multicast-routing proposals.

Each link $l \in L$ has an associated cost $c(l) \geq 0$ that is known by the nodes on each end, and each user i assigns a utility value u_i to receiving the transmission. Note that u_i is known only to user i *a priori*, and hence user i can strategize by reporting any value $v_i \geq 0$ in place of u_i. A *cost-sharing mechanism* determines which users receive the multicast transmission and how much each receiver is charged. We let $x_i \geq 0$ denote how much user i is charged and σ_i denote whether user i receives the transmission; $\sigma_i = 1$ if the user receives the multicast transmission, and $\sigma_i = 0$ otherwise. We use u to denote the input vector $(u_1, u_2, \ldots, u_{|P|})$. The mechanism M is then a pair of functions $M(u) = (x(u), \sigma(u))$. The practical feasibility of deploying the mechanism on the Internet depends on the network complexity of computing the functions $x(u)$ and $\sigma(u)$. It is important to note that both the inputs and outputs of these functions are distributed throughout the network; that is, each user inputs his u_i from his network location, and the outputs $x_i(u)$ and $\sigma_i(u)$ must be delivered to him at that location.

The *receiver set* for a given input vector is $R(u) = \{i \mid \sigma_i = 1\}$. A user's individual *welfare* is given by $w_i = \sigma_i u_i - x_i$. The cost of the tree $T(R)$ reaching a set of receivers R is $c(T(R))$, and the overall welfare, or *net worth*, is $NW(R) =$

[1] In particular, these recent papers study the *network complexity* of the problem. This measure of complexity takes into account both local computational costs and several aspects of communication costs; see [6] for a thorough introduction to distributed algorithmic mechanism design.

$u_R - c(T(R))$, where $u_R = \sum_{i \in R} u_i$ and $c(T(R)) = \sum_{l \in T(R)} c(l)$. The overall welfare measures the total benefit of providing the multicast transmission (the sum of the utilities minus the total cost).

Our goal is to explore the relationship between incentives and computational complexity, but, before we do so, we first comment on several aspects of the model. The cost model we employ is a poor reflection of reality, in that transmission costs are not per-link; current network-pricing schemes typically only involve usage-based or flat-rate access fees, and the true underlying costs of network usage, though hard to determine, involve small incremental costs (*i.e.*, sending additional packets is essentially free) and large fixed costs (*i.e.*, installing a link is expensive). However, we are not aware of a well-validated alternative cost model, and the per-link cost structure is intuitively appealing, relatively tractable, and widely used.

There are certainly cases, such as the high-bandwidth broadcast of a long-lived event such as a concert or movie, in which the bandwidth required by the transmission is much greater than that required by a centralized cost-sharing mechanism (*i.e.*, sending all the link costs and utility values to a central site at which the receiver set and cost shares could be computed). For these cases, our feasibility concerns would be moot. However, Internet protocols are designed to be general-purpose; what we address here is the design of a protocol that would share multicast costs for a wide variety of uses, not just long-lived and high-bandwidth events. Thus, the fact that there are scenarios (*e.g.*, the transmission of a shuttle mission, as explained below) in which our feasibility concerns are relevant is sufficient motivation; they need not be relevant in all scenarios.

In comparing the bandwidth required for transmission to the bandwidth required for the cost-sharing mechanism, one must consider several factors. First, and most obvious, is the transmission rate b of the application. For large multicast groups, it will be quite likely that there will be at least one user connected to the Internet by a slow modem. Because the multicast rate must be chosen to accommodate the slowest user, one can't assume that b will be large. Second, the bandwidth consumed on any particular link by centralized cost sharing mechanisms scales linearly with the number of users $p = |P|$, but the multicast's usage of the link is independent of the number of users. Third, one must consider the time increment Δ over which the cost accounting is done. For some events, such as a movie, it would be appropriate to calculate the cost shares once (at the beginning of the transmission) and not allow users to join after the transmission has started. For other events, such as the transmission of a shuttle mission, users would come and go during the course of the transmission. To share costs accurately in such cases, the time increment Δ must be fairly short. The accounting bandwidth on a single link scales roughly as p, which must be compared to the bandwidth Δb used over a single accounting interval. Although small multicast groups with large Δ and b could easily use a centralized mechanism, large multicast groups with small Δ and b could not.

We have assumed that budget-balanced cost sharing, where the sum of the charges exactly covers the total incurred cost, is the goal of the charging mechanism. If the charging mechanism were being designed by a monopoly network operator, then one might expect the goal to be maximizing revenue. There have been some recent investigations of revenue-maximizing charging schemes for multicast (see, *e.g.*, [7]), but here we assume, as in [9,14,5,2,1], that the charging mechanism is decided by society at large (*e.g.*, through standards bodies) or through competition. Competing network providers could not charge more than their real costs (or otherwise their prices would be undercut) nor less than their real costs (or else they would lose money), and so budget balance is a reasonable goal in such a case. For some applications, such as big-budget movies, the bandwidth costs will be insignificant compared to the cost of the content, and then different charging schemes will be needed, but for low-budget or free content (*e.g.*, teleconferences) budget-balanced cost-sharing is appropriate.

Lastly, in our model it is the *users* who are selfish. The routers (represented by tree nodes), links, and other network-infrastructure components are obedient. Thus, the cost-sharing algorithm does not know the individual utilities u_i, and so users could lie about them, but once they report them to the network infrastructure (*e.g.*, by sending them to the nearest router or accounting node), the algorithms for computing $x(u)$ and $\sigma(u)$ can be reliably executed by the network. Ours is the simplest possible strategic model for the distributed algorithmic mechanism-design problem of multicast cost sharing, but, even in this simplest case, determining the inherent network complexity of the problem is non-trivial. Alternative strategic models (*e.g.*, ones in which the routers are selfish, and their strategic goals may be aligned or at odds with those of their resident users) may also present interesting distributed algorithmic mechanism-design challenges. Preliminary work along these lines is reported in [15].

1.2 Statement of Results

In order to state our results more precisely, we need additional notation and terminology.

A *strategyproof* cost-sharing mechanism is one that satisfies the property that $w_i(u) \geq w_i(u|^i\mu_i)$, for all u, i, and μ_i. (Here, $(u|^i\mu_i)_j = u_j$, for $j \neq i$, and $(u|^i\mu_i)_i = \mu_i$. In other words, $u|^i\mu_i$ is the utility profile obtained by replacing u_i by μ_i in u.) Strategyproofness does not preclude the possibility of a group of users colluding to improve their individual welfares. Any reported utility profile v can be considered a group strategy for any group $S \supseteq \{i \mid v_i \neq u_i\}$. A mechanism M is *group-strategyproof* (GSP) if there is no group strategy such that at least one member of the strategizing group improves his welfare while the rest of the members do not reduce their welfare. In other words, if M is GSP, the following property holds for all u, v, and $S \supseteq \{i|u_i \neq v_i\}$: *either* $w_i(v) = w_i(u)$, $\forall i \in S$, or $\exists i \in S$ such that $w_i(v) < w_i(u)$.

In general, we only consider mechanisms that satisfy four natural requirements[2]:

No Positive Transfers (NPT): $x_i(u) \geq 0$; in other words, the mechanism cannot *pay* receivers to receive the transmission.

[2] The one exception is Section 4, in which we do not assume SYM; that section contains an impossibility result, and so not making this assumption only makes the section stronger.

Voluntary Participation (VP): $w_i(u) \geq 0$; this implies that users are not charged if they do not receive the transmission and that users who do receive the transmission are not charged more than their reported utilities.
Consumer Sovereignty (CS): For given $T(P)$ [3] and link costs $c(\cdot)$, there exists some κ such that $\sigma_i(u) = 1$ if $u_i \geq \kappa$; this condition ensures that the network cannot exclude any agent who is willing to pay a sufficiently large amount, regardless of other agents' utilities.
Symmetry[4] (SYM): If i and j are at the same node or are at different nodes separated by a zero-cost path, and $u_i = u_j$, then $x_i = x_j$.

In addition to these basic requirements, there are certain other desirable properties that one could expect a cost-sharing mechanism to possess. A cost-sharing mechanism is said to be *efficient* if it maximizes the overall welfare, and it is said to be *budget-balanced* if the revenue raised from the receivers covers the cost of the transmission exactly. It is a classical result in game theory [8] that a strategyproof cost-sharing mechanism that satisfies NPT, VP, and CS cannot be both budget-balanced and efficient. Moulin and Shenker [14] have shown that there is only one strategyproof, efficient mechanism, called *marginal cost* (MC) that satisfies NPT, VP, and CS. They have also shown that, while there are many GSP, budget-balanced mechanisms that satisfy NPT, VP, and CS, the most natural one to consider is the *Shapley value* (SH), defined in Section 2 below, because it minimizes the worst-case efficiency loss.

Both MC and SH also satisfy the SYM property. The *egalitarian* (EG) mechanism of Dutta and Ray [3] is another well studied GSP, budget-balanced mechanism that satisfies the four basic requirements. Jain and Vazirani [11] present a novel family of GSP, approximately budget-balanced mechanisms[5] that satisfy NPT, VP, and CS. Each mechanism in the family is defined by its underlying cost-sharing function, and the resulting mechanism satisfies the SYM property whenever the underlying function satisfies it. We use the notation JV to refer to the members of the Jain-Vazirani family that satisfy SYM.

It is easy to see (and is noted in [5]) that both MC and SH are polynomial-time computable by centralized algorithms. Furthermore, there is a distributed algorithm given in [5] that computes MC using only two short messages per link and two simple calculations per node. By contrast, [5] notes that the obvious algorithm that computes SH requires $\Omega(|P| \cdot |N|)$ messages in the worst case and shows that, for a restricted class of algorithms (called "linear distributed algorithms"), there is an infinite set of instances with $|P| = O(|N|)$ that require $\Omega(|N|^2)$ messages. Jain and Vazirani [11] give centralized, polynomial-time algo-

[3] For brevity, we often use $T(P)$ to denote four components of a multicast cost-sharing problem instance: the node-set N, the link-set L, the locations of the agents, and the multicast-source location α_s.

[4] This straightforward definition is less restrictive than the one given by Moulin and Shenker [14]. The SH, JV, and EG mechanisms that we use as examples satisfy the more stringent definition of symmetry in [14] as well.

[5] The mechanisms in [11] actually satisfy a more stringent definition of approximate budget balance than we use; thus, our network-complexity lower bounds apply to them *a fortiori*.

rithms to compute the approximately budget-balanced mechanisms in the class JV.

In this paper, we show that:

- Any distributed algorithm, deterministic or randomized, that computes a budget-balanced, GSP multicast cost-sharing mechanism must send $\Omega(|P|)$ bits over linearly many links in the worst case. This lower bound applies, in particular, to the SH and EG mechanisms.
- Any distributed algorithm, deterministic or randomized, that computes an approximately budget-balanced, GSP multicast cost-sharing mechanism must send $\Omega(\log(|P|))$ bits over linearly many links in the worst case. This lower bound applies, in particular, to the SH, EG, and JV mechanisms.

In order to prove the first of these lower bounds (*i.e.*, the one for exact budget balance), we first prove a lower bound that holds for all mechanisms that correspond to *strictly cross-monotonic* cost-sharing functions. Cross-monotonicity, a technical property defined precisely in Section 2, means roughly that the cost share attributed to any particular receiver cannot increase as the receiver set grows; the SH and EG cost-sharing functions for a broad class of multicast trees are examples of strictly cross-monotonic functions but not the only examples. Our lower bound on the network complexity of strictly cross-monotonic mechanisms may be applicable to problems other than multicast.

Finally, we prove the following generalization of a classical result in game theory [8]:

- There is no strategyproof multicast cost-sharing mechanism satisfying NPT, VP, and CS that is both approximately efficient and approximately budget-balanced.

In what follows, most proofs and technical details are omitted because of space limitations. They can be found in our journal submission [4].

2 Exact Submodular Cost Sharing

In this section, we prove a basic communication-complexity lower bound that applies to the distributed computation of many submodular cost-sharing mechanisms. We first prove this lower bound for all mechanisms that satisfy "strict cross-monotonicity" as well as the four basic properties discussed in Section 1. We then show that, whenever the underlying cost function is strictly subadditive, the resulting Shapley-value mechanism is strictly cross-monotonic and hence has poor network complexity. Finally, we discuss the special case of multicast cost sharing and describe very general conditions under which the multicast cost will be strictly subadditive. In particular, we present an infinite family of instances that have strictly subadditive costs and show that any cost-sharing mechanism that satisfies the four basic requirements must have poor network complexity on these instances.

Consider the general situation in which we want a mechanism to allow the users to share the cost of a common service. We restrict our attention to the case of binary preferences: User i is either "included," by which he attains utility u_i, or he is "excluded" from the service, giving him 0 utility. A mechanism can use the utility vector u as input to compute a set $R(u)$ of users who receive the service and a payment vector $x(u)$. Further, suppose that the cost of serving a set $S \subseteq P$ of the users is given by $C(S)$. This cost function is called *submodular* if, for all $S, T \subseteq P$, it satisfies: $C(S \cup T) + C(S \cap T) \leq C(S) + C(T)$.

Submodularity is often used to model economies of scale, in which the marginal costs decrease as the serviced set grows. One example of a submodular cost function is the one presented in Section 1, where the cost of delivering a multicast to a set R of receivers is the sum of the link costs in the smallest subtree of the universal tree that includes all locations of users in R.

Moulin and Shenker [13,14] have shown that any mechanism for submodular cost sharing that satisfies budget-balance, GSP, VP, and NPT must belong to the class of *cross-monotonic cost-sharing mechanisms*. A mechanism in this class is completely characterized by its set of cost-sharing functions $g = \{g_i : 2^P \rightarrow \Re_{\geq 0}\}$. Here $g_i(S)$ is the cost that g attributes to user i if the receiver set is S. For brevity, we will refer to $g = \{g_i\}$ as a "cost-sharing function," rather than a set of cost-sharing functions. We say that g is *cross-monotonic* if, $\forall i \in S, \forall T \subseteq P,\ g_i(S \cup T) \leq g_i(S)$. In addition, we require that $g_i(S) \geq 0$ and, $\forall j \notin S,\ g_j(S) = 0$. Then, the corresponding cross-monotonic mechanism $M_g = (\sigma(u), x(u))$ is defined as follows: The receiver set $R(u)$ is the unique largest set S for which $g_i(S) \leq u_i$, for all i. This is well defined, because, if sets S and T each satisfy this property, then cross-monotonicity implies that $S \cup T$ satisfies it. The cost shares are then set at $x_i(u) = g_i(R(u))$.

There is a natural iterative algorithm to compute a cross-monotonic cost-sharing mechanism [14,5]: Start by assuming the receiver set $R^0 = P$, and compute the resulting cost shares $x_i^0 = g_i(R^0)$. Then drop out any user j such that $u_j < x_j^0$; call the set of remaining users R^1. The cost shares of other users may have increased, so we need to compute the new cost shares $x_i^1 = g_i(R^1)$ and iterate. This process ultimately converges, terminating with the receiver set $R(u)$ and the final cost shares $x_i(u)$.

Now, we consider a subclass of the cross-monotonic mechanisms:

Definition 1 *A cross-monotonic cost-sharing function $g = \{g_i : 2^P \rightarrow \Re_{\geq 0}\}$ is called* **strictly cross-monotonic** *if, for all $S \subset P, i \in S$, and $j \notin S$, $g_i(S \cup j) < g_i(S)$. The corresponding mechanism M_g is called a strictly cross-monotonic mechanism.*

We now prove a lower bound on the communication complexity of strictly cross-monotonic cost-sharing mechanisms. Our proof is a reduction from the *set disjointness* problem: Consider a network consisting of two nodes A and B, separated by a link l. Node A has a set $S_1 \subseteq \{1, 2, \ldots, r\}$, node B has another set $S_2 \subseteq \{1, 2, \ldots, r\}$, and one must determine whether the sets S_1 and S_2 are disjoint. It is known that any deterministic or randomized algorithm to solve this problem must send $\Omega(r)$ bits between A and B. (Proofs of this and other basic results in communication complexity can be found in [12].)

Theorem 1 *Suppose M_g is a strictly cross-monotonic mechanism corresponding to a cost-sharing function g and satisfying VP, CS, and NPT. Further, suppose that the mechanism must be computed in a network in which a link (or set of links) l is a cut and there are $\Omega(|P|)$ users on each side of l. Then, any deter-*

ministic or randomized algorithm to compute M_g must send $\Omega(|P|)$ bits across l in the worst case.

Proof. For simplicity, assume that the network consists of two nodes A and B connected by one link l and that there are $r = |P|/2$ users at each of the two nodes. (The proof of the more general case is identical.) Arbitrarily order the users at each node. We can now call the users $a_1, a_2, \ldots, a_r$ and $b_1, b_2, \ldots, b_r$. Because the mechanism M_g is strictly cross-monotonic, we can find a real value $d > 0$ such that, for all $S \subset P, i \in S, j \notin S$, $g_i(S \cup j) < g_i(S) - d$.

For each user $i \in P$, we will define two possible utility values t_i^L and t_i^H. For $k = 1, 2, \cdots, \frac{|P|}{2}$,

$$t_{a_k}^H = g_{a_k}(\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}), \qquad t_{a_k}^L = t_{a_k}^H - d$$

$$t_{b_k}^H = g_{b_k}(\{a_1, a_2, \ldots, a_k, b_1, b_2, \ldots, b_k\}), \qquad t_{b_k}^L = t_{b_k}^H - d$$

Now, we show how to reduce from the set disjointness problem to the mechanism M_g. Node A gets a subset $S_1 \subseteq \{1, \ldots, r\}$ and constructs a utility vector u for the users at A, defined by, $\forall i \in S_1$, $u_{a_i} = t_{a_i}^H$, and, $\forall i \notin S_1$, $u_{a_i} = t_{a_i}^L$. Similarly, node B is given set S_2 and constructs a utility vector v for the users at B, defined by, $\forall i \in S_2$, $v_{b_i} = t_{b_i}^H$, and, $\forall i \notin S_2$, $v_{b_i} = t_{b_i}^L$.

They now run mechanism M_g on input (u, v) and check whether the receiver set $R_g(u, v)$ is empty.

Claim: $R_g(u, v)$ is empty iff S_1 and S_2 are disjoint.

Proof of claim: To show the "if" direction, we can simulate the iterative algorithm to compute the receiver set. We start with $R = P$. Then, because S_1 and S_2 are disjoint, either $r \notin S_1$ or $r \notin S_2$. Assume, without loss of generality, that $r \notin S_1$. Now, $u_{a_r} = t_{a_r}^L < g_{a_r}(R)$, and hence a_r must drop out of the receiver set R. But now, because of strict cross-monotonicity, it follows that $g_{b_r}(P - \{a_r\}) > g_{b_r}(P) + d > t_{b_r}^H$, and so b_r must also drop out of the receiver set. Repeating this argument for $r-1, r-2, \ldots, 1$, we can show that the receiver set must be empty.

To show the "only if" direction, assume that $i \in S_1 \cap S_2$. Then, let $T = \{a_1, \ldots, a_i, b_1, \ldots, b_i\}$. $u_{a_i} = t_{a_i}^H = g_{a_i}(T)$, and $v_{a_i} = t_{b_i}^H = g_{b_i}(T)$. Further, for all $j < i$, it follows from strict cross-monotonicity that $g_{a_j}(T) < t_{a_j}^L \leq u_{a_j}$, and $g_{b_j}(T) < t_{b_j}^L \leq v_{b_j}(T)$. Thus, the receiver set $R_g(u, v) \supseteq T$, and hence it is nonempty. □

Theorem 1 follows from this claim and the communication complexity of set disjointness. □

2.1 Strictly Subadditive Cost Functions

In this section, we show that, for a class of submodular cost functions, the Shapley-value mechanism (which is perhaps the most compelling mechanism from an economic point of view) is strictly cross-monotonic and hence has poor network complexity. We also show that this is not a property peculiar to the Shapley-value mechanism alone; for these cost functions, the poor network complexity holds for a large class of mechanisms.

Theorem 1 provides a sufficient condition, strict cross-monotonicity, for a mechanism to have poor network complexity. However, for some submodular cost functions, it is possible that no mechanism satisfies this condition: If the costs are additive, *i.e.*, if the cost of serving a set S is exactly the sum of the costs of serving each of its members individually, then there is a unique mechanism satisfying the basic properties. This mechanism is defined by $R(u) = \{i|u_i \geq C(\{i\})\}$ and $x_i(u) = C(\{i\})$ if $i \in R(u)$, and $x_i(u) = 0$ otherwise. This mechanism is very easy to compute, either centrally or in a distributed manner, because there is no interaction among the users' utilities; in essence, we have $|P|$ independent local computations to perform.

We need to exclude these trivial cost functions in order to prove general lower bounds for a class of submodular functions. This leads us to consider submodular cost functions that are *strictly subadditive*: $\forall S \subseteq P$, $S \neq \emptyset$, and, $\forall i \in P$, $C(S \cup \{i\}) < C(S) + C(\{i\})$.

For a given cost function C, there may be many $g = \{g_i\}$ for which the corresponding mechanism M_g satisfies the basic properties NPT, VP, CS, and SYM. However, Moulin and Shenker [13,14] have shown that, for any given submodular cost function, the cross-monotonic mechanism that minimizes the worst-case efficiency loss is the Shapley-value mechanism (SH). This is a cross-monotonic cost-sharing mechanism corresponding to a function g^{SH}, defined by:

$$\forall S \subseteq P \quad \forall i \in S, \qquad g_i^{SH}(S) = \sum_{R \subseteq S - \{i\}} \frac{|R|!(|S| - |R| - 1)!}{|S|!}[C(R \cup \{i\}) - C(R)]$$

The SH mechanism is therefore a natural mechanism to choose for submodular cost sharing. The following lemma shows that this mechanism has poor network complexity.

Lemma 1. *The Shapley-value mechanism for a strictly subadditive cost function is strictly cross-monotonic.*

Corollary 1. *For a strictly subadditive cost function, any algorithm (deterministic or randomized) that computes the SH mechanism in a network must communicate $\Omega(|P|)$ bits across any cut that has $\Theta(|P|)$ users on each side of the cut.* □

Note that the network may consist of a root node α_s with no resident users, a node A with $\frac{|P|}{2}$ resident users, another node B with $\frac{|P|}{2}$ resident users, a link from α_s to A, and a path from A to B consisting of $|N| - 3$ nodes, each with no resident users. Each link in the path from A to B is a cut with $\Theta(|P|)$ users on each side, and thus $\Omega(|P|)$ bits must be sent across linearly many links. In what follows, we call these the *path instances*.

2.2 Multicast Cost Sharing

We now return to the special case of multicast cost sharing. Recall that the cost function associated with an instance of the multicast cost-sharing problem is determined by the structure of the universal multicast tree T, the link costs, and the locations of the users in the tree; so the cost $C(S)$ of serving user set $S \subseteq P$ is $\sum_{l \in T(S)} c(l)$, where $T(S)$ is the smallest subtree of T that includes all nodes at which users in S reside. It is not hard to show that there are many instances that give rise to strictly subadditive functions C. In fact, we have the following lemma:

Lemma 2. *Consider any instance of multicast cost sharing in which, for any two potential receivers i and j, there exists a link $l \in T(\{i\}) \cap T(\{j\})$ such that $c(l) > 0$. The cost function associated with this instance is strictly subadditive.*

For example, whenever the source of the multicast has only one link from it, and this link has non-zero cost, the associated cost function is strictly subadditive. One such family of instances are path instances with cost C on the link from α_s to A and cost 0 on all the other links.

It follows immediately from Corollary 1 that the Shapley-value mechanism for this family of trees requires $\Omega(|P|)$ bits of communication across linearly many links. In addition, we now show that any mechanism that satisfies the basic properties must be identical to the SH mechanism on these path instances; thus, the lower bound extends to all such mechanisms.

Lemma 3. *Consider multicast cost-sharing problems induced by path instances. Let M_g be a cross-monotonic cost-sharing mechanism that satisfies SYM, corresponding to a cost-sharing function $g = \{g_i\}$. Then, g (and M_g) are completely determined on these instances by $\forall S \subseteq P, \forall i \in S, \;\; g_i(S) = \frac{C}{|S|}$.*

It follows from Lemma 3 that any such mechanism must be strictly cross-monotonic on this family of instances. Thus, Theorem 1 and Lemma 3 imply the following lower bound for multicast cost sharing.

Theorem 2 *Any distributed algorithm, deterministic or randomized, that computes a budget-balanced, GSP multicast cost-sharing mechanism exactly must send $\Omega(|P|)$ bits over linearly many links in the worst case.* □

Note that this lower bound applies to the EG mechanism for multicast cost-sharing defined in Section 1.

3 Network Complexity of Approximately Budget-Balanced Mechanisms

For our purposes in this section, a *κ-approximately budget-balanced mechanism*, where $\kappa > 1$ is a constant, is a mechanism (σ, x) with the following properties: VP, NPT, CS, SYM, and

$$\forall c(\cdot), T(P), \text{ and } u : (1/\kappa) \cdot c(T(R(u))) \leq \sum_{i \in R(u)} x_i(u) \leq \kappa \cdot c(T(R(u))).$$

An *approximately budget-balanced mechanism* is one that is κ-approximately budget-balanced for some $\kappa > 1$.

Theorem 3 *Any distributed algorithm, deterministic or randomized, that computes a κ-approximately budget-balanced, GSP multicast cost-sharing mechanism, where $\kappa \leq \sqrt{2} - \epsilon$, for some fixed $\epsilon > 0$, must send $\Omega(\frac{\log |P|}{\log \kappa})$ bits of communication over linearly many links in the worst case.*

Once again, the worst-case instances include the path instances defined in Section 2. For the proof of Theorem 3, please refer to our journal submission [4]. For a general discussion of what it means to "approximate a mechanism," see, *e.g.*, [16,2,6].

4 An Impossibility Result for Approximate Budget-Balance and Approximate Efficiency

In this section, we do not assume that the cost-sharing mechanisms have the SYM property; the impossibility result that we present here does not require this assumption. Furthermore, this result only requires the mechanism to be strategyproof, not GSP as in Section 3.

We first review the definition of the MC mechanism, which was shown by Moulin and Shenker [14] to be the only efficient mechanism that satisfies VP, NPT, and CS. Given an input utility profile u, the MC receiver set is the unique *largest efficient set* of users. To compute it, as shown in [5], one recursively computes the *welfare* (also known as *net worth* or *efficiency*) of each node $\beta \in N$:

$$W(\beta) = \left(\sum_{\substack{\gamma \in Ch(\beta) \\ W(\gamma) \geq 0}} W(\gamma) \right) - c(l) + \sum_{i \in Res(\beta)} u_i \ ,$$

where $Ch(\beta)$ is the set of children of β in the tree, $Res(\beta)$ is the set of users resident at β, and $c(l)$ is the cost of the link connecting β to its parent node. Then, the largest efficient set $R(u)$ is the set of all users i such that every node on the path from i to the root α_S has nonnegative welfare. The *total efficiency* is $\mathrm{N}W(R(u)) = W(\alpha_S)$.

Let $X(i,u)$ be the node with minimum welfare value in the path from i to its root in its partition. Then, the cost share $x_i(u)$ of user i is defined as

$$\begin{aligned} x_i(u) &= \max(0, u_i - W(X(i,u))) \quad && \forall i \in R(u) \\ x_i(u) &= 0 && \forall i \notin R(u) \end{aligned}$$

If multiple nodes on the path have the same welfare value, we let $X(i,u)$ be the one nearest to i.

By a *γ-approximately efficient mechanism*, where $0 < \gamma < 1$, we mean one that always achieves total efficiency that is at least γ times the total efficiency achieved by MC.

Theorem 4 *A strategyproof mechanism for multicast cost sharing that satisfies the basic requirements of NPT, VP, and CS cannot achieve both γ-approximate efficiency and κ-approximate budget-balance for any pair of constants κ and γ.*

Both the proof of Theorem 4 and an explicit family of instances on which no strategyproof mechanism satisfying the basic requirements achieves approximate efficiency and approximate budget balance can be found in our journal submission [4].

References

1. Adler, M. and Rubenstein, D. (2002). "Pricing Multicast in More Practical Network Models," in *Proceedings of the 13th Symposium on Discrete Algorithms*, pp. 981–990, ACM Press/SIAM, New York/Philadelphia.
2. Archer, A., Feigenbaum, J., Krishnamurthy, A., Sami, R., and Shenker, S. (2002). "Approximation and Collusion in Multicast Cost Sharing," submitted. `http://www.cs.yale.edu/homes/jf/AFKSS.ps`
3. Dutta, B. and Ray, D. (1989). "A concept of egalitarianism under participation constraints," *Econometrica* **57**, pp. 615-635.
4. Feigenbaum, J., Krishnamurthy, A., Sami, R., and Shenker, S. (2002). "Hardness Results for Multicast Cost Sharing," submitted. `http://www.cs.yale.edu/homes/jf/FKSS2.ps`.
5. Feigenbaum, J., Papadimitriou, C., and Shenker, S. (2001). "Sharing the cost of multicast transmissions," *Journal of Computer and System Sciences* **63**, pp. 21–41.
6. Feigenbaum, J. and Shenker, S. (2002). "Distributed Algorithmic Mechanism Design: Recent Results and Future Directions," in *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, pp. 1–13, ACM Press, New York.
7. Fiat, A., Goldberg, A., Hartline, J., and Karlin, A. (2002). "Competitive Generalized Auctions," in *Proceedings of the 34th Symposium on the Theory of Computing*, pp. 72–81, ACM Press, New York.
8. Green, J. and Laffont, J-J. (1979). **Incentives in Public Decision Making**, North Holland, Amsterdam.
9. Herzog, S., Shenker, S., and Estrin, D. (1997). "Sharing the 'cost' of multicast trees: An axiomatic analysis," *IEEE/ACM Transactions on Networking* **5**, pp. 847–860.
10. Jackson, M. (2001). "A Crash Course in Implementation Theory," *Social Choice and Welfare* **18**, pp. 655–708.
11. Jain, K. and Vazirani, V. (2001). "Applications of approximation to cooperative games," in *Proceedings of the 33rd Symposium on the Theory of Computing*, pp. 364–372, ACM Press, New York.
12. Kushilevitz, E. and Nisan, N. (1997). **Communication Complexity**, Cambridge University Press, Cambridge.
13. Moulin, H. (1999). "Incremental cost sharing: characterization by strategyproofness," *Social Choice and Welfare* **16**, pp. 279–320.
14. Moulin, H. and Shenker, S. (2001). "Strategyproof Sharing of Submodular Costs: Budget Balance Versus Efficiency," *Economic Theory* **18**, pp. 511–533.
15. Mitchell, J. and Teague, V. (2002). Private communication.
16. Nisan, N. and Ronen, A. (2000). "Computationally feasible VCG mechanisms," in *Proceedings of the 2nd Conference on Electronic Commerce (EC-00)*, pp. 242–252, ACM Press, New York.
17. Nisan, N. and Ronen, A. (2001). "Algorithmic mechanism design," *Games and Economic Behavior* **35**, pp. 166–196.

How to Compose Presburger-Accelerations: Applications to Broadcast Protocols

Alain Finkel and Jérôme Leroux

Laboratoire Spécification et Vérification
CNRS UMR 8643 & ENS de Cachan
61 av. du Président Wilson
94235 Cachan cedex, France
{finkel,leroux}@lsv.ens-cachan.fr

Abstract. *Finite linear systems* are finite sets of linear functions whose guards are defined by Presburger formulas, and whose the squares matrices associated generate a finite multiplicative monoid. We prove that for finite linear systems, the accelerations of sequences of transitions always produce an effective Presburger-definable relation. We then show how to choose the good sequences of length n whose number is *polynomial* in n although the total number of sequences of length n is exponential in n. We implement these theoretical results in the tool `FAST` [FAS] (Fast Acceleration of Symbolic Transition systems). `FAST` computes in few seconds the minimal deterministic finite automata that represent the reachability sets of 8 well-known broadcast protocols.

Keywords: Presburger model checking, verification, infinite-state systems, reachability set, acceleration.

1 Introduction

Reachability is a central problem in verification. Let us recall that verification of safety properties can be reduced to reachability of a given state from a set of initial states. For some particular models like lossy channel systems [AJ96], [AAB99] automata with stacks [FWW97], [BEM97], reactive FIFO automata [SFRC99], flat counters automata ([CJ98] and reset/transfer 2-counters automata [FS00b], [FS00a], it is possible to test reachability from a regular set of initial states because the reachability set (or the set of all predecessors) is regular and is effectively computable. But, these "decidable" models are often too restricted because we may wish to use counters with no constraints on guards and actions, as for example, broadcast protocols with parameterized initial set of states [EN98], [EFM99], [Del00]. Hence, our desired model must be able to *easily* simulate counters automata and as an unavoidable consequence, most of reachability problems will become undecidable.

We make the observation that almost all systems with integer variables can be seen as a *finite set of linear functions* $f(x) = M.x + v : D \subseteq \mathbb{Z}^m \to \mathbb{Z}^m$

having some *definition domains* D. For being able to effectively handle these functions f, we will suppose that their definition domains D (also called guards) are Presburger-definable; such systems are now called *linear systems*. It is clear that the reachability problem is undecidable for linear systems even of dimension 3 (by using a Post problem reduction [FB95]).

We call *Presburger Model Checking* the Regular Model Checking [BJNT00], [FO97b], [BF00], [BGP97] in which Presburger formulas (instead of regular languages) are used as a symbolic representation of infinite set of states. Presburger formulas enjoy good properties because, as regular sets, they are closed under union, intersection and negation; moreover, the satisfiability and the validity are both decidable [GS66].

We say that we *accelerate a sequence of transitions labeled by* σ from a set S of states when we symbolically compute the infinite union of all $\sigma^n(S)$; this set is also called the σ^*-acceleration set. We will compute and use accelerations of different sequences, to speed up the reachability set construction and help its termination. Accelerations are also called meta-transitions in [BW94], [Boi98] or exact widening in the field of abstract interpretation.

Boigelot's Model and Results

B. Boigelot and P. Wolper in [BW94], [WB98] and [Boi98] consider a *unique* linear function f whose guard is of the form $Ax \leq b$. They gave two decidable technical conditions [B1] and [B2] over a linear function f (called here a Boigelot's function) which insures the reachability set $f^*(S)$ (i.e., the acceleration of f) to be effectively Presburger-definable when S is Presburger-definable. Conditions [B1] and [B2] become necessary and sufficient when we consider totally-defined linear functions (it means guards are equal to $\mathbb{Z}^m$).

Our Contributions

- **Finite linear systems enable composition of accelerations**
  The starting point of our research has been to find out conditions to be able to make acceleration of any composition of functions of a linear system. We remarked that the two technical conditions [B1] and [B2] exactly correspond to the finiteness of the multiplicative monoid generated by the square matrix M associated f. Hence, it is natural to consider the class of linear systems whose the monoid generated by all the square matrices M, is finite (called *finite linear systems*). This condition is decidable [Jac78],[MS77]. We prove that acceleration of f_σ is possible for any sequence σ.
- **How to find out the good accelerations?**
  The second step to really use accelerations is to compute how many different accelerations are necessary for calculating the reachability relation or the reachability set. We prove that from the exponential number in k of different accelerations of words in $\Sigma^{\leq k}$, we can extract a polynomial number of "good accelerations". Let f and g be two functions, under conditions on f and g, we replace acceleration of f and g by acceleration of the unique function $f \vee g$

whose the definition domain is equal to the union of the definition domains of f and g. To do so, we use the property that the union of two Presburger-guards is still a Presburger-guard (remark that the guards used by Boigelot are not stable by union because they are equal to convex sets of the form $A.x \leq b$). Moreover, the Theorem 4 shows us that for a subclass of totally-defined linear systems, we have $k \leq |\mathcal{M}_{\boldsymbol{L}}|$ ($\mathcal{M}_{\boldsymbol{L}}$ is the finite multiplicative monoid associated to a linear system $\boldsymbol{L}$).

– **The tool `FAST`**
A tool named `FAST` (Fast Acceleration of Symbolic Transition systems) implements the theoretical results of this paper. The current version of `FAST` computes, in few seconds, all the complete reachability sets (represented by finite automata having less than 12 states) of the 8 well-known broadcast protocols (and some other systems known to be difficult to analyze). To the best of our knowledge, it is the first time that a tool computes automatically the complete reachability sets of all the 8 broadcast protocols (and not only the set of all predecessors of an upward closed set of bad states).

Related Works on Acceleration

H. Comon and Y. Jurski analyze a relational model based on multiple counters automata in [CJ98] but their guards cannot express set of integers satisfying $x + y = z$. On the other hand, actions are relations (they are not restricted to be functions). Hence, their model is not comparable with finite linear systems. A. Finkel and G. Sutre [Sut00], [FS00b] study 2-counters automata with reset/transfer/+1/-1 actions and guards of the form $x_i \geq c$ and $x_i = 0$ (zero test). They prove that acceleration of cycles leads to the effective computation of the Presburger-definable reachability set.
[FS00b] considers acceleration from a theoretical point of view with applications to 2-counters automata and to FIFO channel systems. [AAB00] guesses the result of the infinite iteration of a cycle, in a hybrid system, and verify whether the guess is correct or not. [Rev90], [FO97b], [FO97a] and [BF99] automatically accelerate a given non-elementary cycle. [PS00] attacks the acceleration techniques using formulas in a variant of $WS1S$ for the symbolic representation of sets of states: the authors experiment their tool on examples but no decidability result is given about the convergence in general. Among a lot of papers on some what we called the Presburger Model Checking, we can cite [BF00] and [BGP97] which both also use the tool `MONA` [MON]. For FIFO channels systems, acceleration of loops have been recently studied in [FPS00] with SLRE, in [BGWW97] with QDD and in [BH99] with CQDD; for lossy FIFO channels systems, acceleration of loops and SRE have been studied in [AAB99].

Plan of the Paper

Section 2 and 3 intuitively introduce, on a simple example, acceleration of cycles. Then, after recalling the basic notions (labeled transition systems, matrices and functions, Presburger formulas), Section 4 presents the so-called finite linear systems model. The main theoretical result of this section is: the acceleration relation of cycles of a finite linear system is Presburger-definable (Theorem 2). An "if and only if" condition is given for finite totally-defined linear systems. The finiteness of the monoid is decidable and it is EXPTIME for positive finite linear systems. Sections 5 deals with the problem of finding a little number of useful cycles for acceleration. The finiteness of the monoid is used for proving that only a polynomial number of cycles have to be considered for acceleration. Section 6 reports the use of the tool `FAST` (Fast Acceleration of Symbolic Transition systems) specially on broadcast protocols.

2 Preliminaries

A *labeled transition system* LTS is a tuple $LTS = (S, \Sigma, \rightarrow)$ where S is the set of *states*, Σ is a finite alphabet of *actions*, and $\xrightarrow{a} \subseteq S \times S$ is the *transition relation* for the action a. For some letters a_1, ..., a_n, the relation $\xrightarrow{a_1 \cdots a_n}$ is naturally defined by $s \xrightarrow{a_1 \cdots a_n} s'$ if and only if $s \xrightarrow{a_1} \cdots \xrightarrow{a_n} s'$; and for every s, we have $s \xrightarrow{\epsilon} s$ where ϵ is the empty word. A *labeled transition system with initial states* is a couple (LTS, S_0) where LTS is a labeled transition system and $S_0 \subseteq S$ is a set of *initial states*. The *reachability set* of a labeled transition system is $Reach(LTS, S_0) = \{s' \in S; \exists \sigma \in \Sigma^* \; \exists s_0 \in S_0 \; s_0 \xrightarrow{\sigma} s'\}$.

Throughout this paper, $\mathbb{Z}$ is the set of *integers* and $\mathbb{N}$ is the set of *non negative integers*. The set of *square matrices of size* $m \times m$ over the integers $\mathbb{Z}$ is denoted by $\mathcal{M}_m(\mathbb{Z})$ and the set of *column matrices* of size $m \times 1$, also called *vectors*, is written $\mathbb{Z}^m$. Given an integer $1 \leq i \leq m$, v_i is the i-th component of a vector $v \in \mathbb{Z}^m$. The vector x such that $x_i = 1$ and $x_j = 0$ for every $j \neq i$ is written $\mathbf{x}_i$.

Consider a finite set Y of *variables*. The set of *arithmetical terms* over Y, is defined by the grammar $t ::= 0|1|y|t - t|t + t$. The set of *Presburger formulas* over Y is defined by the grammar: $\phi ::= t < t|\neg\phi|\phi \vee \phi|\exists x'.\phi|true$. The subset of variables $x' \in Y$ that appear in a quantification is written X'. The set $X = Y - X'$ is called the set of free variables. A Presburger formula over the set of free variables X is written $\phi(X)$. An *interpretation* v of $\phi(X)$ is a vector in $\mathbb{Z}^X$. We say that v *satisfies* $\phi(X)$ which is written $v \models \phi(X)$ if by replacing any occurrence of $x \in \phi(X)$ by the integer $v(x)$ we obtain a true formula over the integers. The set of vectors satisfying a Presburger formula $\phi(X)$ is written $S(\phi)$. A set $S \subseteq \mathbb{Z}^X$ is said to be *Presburger-definable* if there exists a Presburger formula $\phi(X)$ such that $S = S(\phi)$.

Consider some functions $(f_a)_{a \in \Sigma}$ where Σ is a finite alphabet, the function $f_{a_n} \circ \cdots \circ f_{a_1}$ is written $f_{a_1 \cdots a_n}$ to improve the readability. For the same reason, the square matrix $M_{a_n} \cdots M_{a_1}$ is written $M_{a_1 \cdots a_n}$. The empty word ϵ has a length 0 and by convention the matrix M_ϵ is equal to the *identity matrix* I and the function f_ϵ is equal to the *identity function*.

3 What Is Acceleration?

The reachability set of a labeled transition systems is not finitely computable in general. In fact, assume that the transition system has an infinite number of reachable states, then an algorithm that computes one reachable state at one step never terminates. To handle such infinite-state labeled transition systems, we must find out a method that computes, in a finite number of steps, an infinite number of reachable states. To do so, we accelerate sequences of transitions (also called meta-transitions [BW94] or in the field of abstract interpretation, called exact widening)

Definition 1 (σ^*-acceleration). *The relation $s \xrightarrow{\sigma^*} s'$ for a sequence of transitions σ in a LTS is called the* σ^*-acceleration relation. *The set of reachable states by a σ^*-acceleration relation is called the* σ^*-acceleration set. *When there is no ambiguity, we just say σ^*-acceleration.*

We present this method here on a simple example: the Producer-Consumer algorithm [BGP97] with a buffer of size a represented in Fig. 1. It contains 3 events corresponding to a write-action e_W and two read-actions e_{R_1} and e_{R_2}.

Data Variables: i, b, o_1, o_2: integer
Constants: a: integer
Initial Condition: $i = a \wedge (b = o_1 = o_2 = 0)$
Events:
e_W **enabled:** $i > 0$ **action:** $i' = i - 1 \wedge b' = b + 1$
e_{R_1} **enabled:** $b > 0$ **action:** $o_1' = o_1 + 1 \wedge b' = b - 1$
e_{R_2} **enabled:** $b > 0$ **action:** $o_2' = o_2 + 1 \wedge b' = b - 1$

Fig. 1. A Producer-Consumer Algorithm (see [BGP97])

The value of the variables and the constant of the algorithm form a vector $(a, i, b, o_1, o_2) \in \mathbb{Z}^5$. The corresponding labeled transition system LTS_{PCA} has thus $S = \mathbb{Z}^5$ as set of states and $\Sigma = \{e_W, e_{R_1}, e_{R_2}\}$ as alphabet. The relations $\xrightarrow{e_W}$, $\xrightarrow{e_{R_1}}$, $\xrightarrow{e_{R_2}}$ are defined in a natural way: for instance $s \xrightarrow{e_W} s'$ if and only if $s_2 > 0$ and $s' = s + (0, -1, 1, 0, 0)$.

Intuitively, one may see that the action e_W can be accelerated from $S_0 = \{(a, i, b, o_1, o_2) \in \mathbb{N}^5; a = i \wedge b = o_1 = o_2 = 0\}$, it means that one may iterate indefinitely e_W^* and then one reaches the set of states $S_1 = \{(a, i, b, o_1, o_2) \in \mathbb{N}^5; a = i + b \wedge o_1 = o_2 = 0\}$. Then one may accelerate $(e_{R_1} e_W)$...

Consider the labeled transition system LTS_{PCA} associated with the Producer-Consumer Algorithm (Fig. 1) where the set of initial states is $S_0 = \{(a, i, b, o_1, o_2) \in \mathbb{N}^5; a = i \wedge b = o_1 = o_2 = 0\}$. By combining different accelerations, we may prove that a state s' is reachable from $s_0 \in S_0$ if and only if $s_0 \xrightarrow{e_W^*} \xrightarrow{(e_{R_1} e_W)^*} \xrightarrow{(e_{R_2} e_W)^*} s'$.From this computation, we can easily deduce the invariant $i = b = 0 \rightarrow a = o_1 + o_2$ which is established in [BGP97].

Such a method for computing the set of reachable states implies that we have found a class of labeled transition systems for which we have a representation of infinite subsets of integers such that

– we can compute a representation of the σ^*-accelerations, and
– we can find out the "good" σ to accelerate which are here e_W, $e_{R_1} e_W$ and $e_{R_2} e_W$.

A subset of $\mathbb{Z}^m$ will be represented by a Presburger formula, the class of finite linear systems will be proved to be a good class of labeled transition systems in the two next sections.

4 How to compute an Acceleration?

We are interested by the computation of accelerations of transitions for labeled transition systems that use tuples of integers to represent their states and *Presburger-linear functions* to represent their actions. A Presburger-linear function f is a function which can be represented by a tuple (M, v, ϕ) where $M \in \mathcal{M}_m(\mathbb{Z})$, $v \in \mathbb{Z}^m$ and $\phi(x_1, \dots, x_m)$ is a Presburger formula, such that $f(s) = M.s + v$ for every $s \models \phi$. Without ambiguity and to improve the readability, such a tuple (M, v, ϕ) and the Presburger-linear function associated with it will be noted by the same letter f. A Presburger-linear function is said to be *totally defined* if it is defined over the whole set $\mathbb{Z}^m$. The totally defined Presburger-linear function associated with a Presburger-linear function $f = (M, v, \phi)$ is written $\bar{f} = (M, v, true)$.

Definition 2 (Linear systems). *A* linear system $\boldsymbol{L}$ *is a triple* $\boldsymbol{L} = (\Sigma, m, f_\Sigma)$ *where* Σ *is a finite alphabet,* $m > 0$ *is a positive integer and* $f_\Sigma = \{f_a = (M_a, v_a, \phi_a); a \in \Sigma\}$ *is a set of Presburger-linear functions into* $\mathbb{Z}^m$.

A labeled transition system can easily be associated with a linear system $\boldsymbol{L} = (\Sigma, m, f_\Sigma)$ by considering $LTS = (\mathbb{Z}^m, \Sigma, \rightarrow)$ where $s \xrightarrow{a} s's$ if and only if $s' = f_a(s)$. The labeled transition system associated with a linear system will be written without ambiguity by the same name.

Definition 3 (The monoid $\mathcal{M}_{\boldsymbol{L}}$). *The multiplicative monoid generated by the matrices* M_a *(for* $a \in \Sigma$*) of a linear system* $\boldsymbol{L}$ *is written* $\mathcal{M}_{\boldsymbol{L}}$*. When* $\boldsymbol{L}$ *is composed of a unique function* $f(s) = M.s + v$*, then the monoid* $\mathcal{M}_{\boldsymbol{L}}$ *is simply written* $< M >$.

The structure of such a monoid has already been studied by Boigelot.

Theorem 1. *[Boi98] For every linear function* $f(s) = M.s + v$ *with a guard defined by* $A.x \leq b$*, where* $A \in \mathbb{Z}^{k \times m}$*,* $b \in \mathbb{Z}^k$*, the set* $f^*(S_0)$ *is Presburger-definable for every Presburger-definable set* S_0 *if there exists an integer* $p > 0$ *such that [B1] and [B2] hold, with:*

– *[B1] the eigenvalues of the matrix M^p belong to $\{0, 1\}$, and,*
– *[B2] the matrix M^p is diagonalizable.*

Moreover, if the guard of f is equal to $\mathbb{Z}^k$, the conditions [B1] and [B2] become necessary and sufficient.

It is easy to see that the two conditions (*[B1]* and *[B2]*) proposed by Boigelot in the previous theorem are equivalent to the fact that the monoid $< M >$ is finite (a proof using elementary algebra results is given in [FL02]). What we want to do, is to be able to compute acceleration of *any* word $\sigma \in \Sigma^*$. Therefore, the following definition seems to be a good one.

Definition 4 (Finite linear systems). *A linear system $\boldsymbol{L}$ is finite if $\mathcal{M}_{\boldsymbol{L}}$ is finite.*

Deciding the finiteness of a linear system is equivalent to deciding the finiteness of a multiplicative monoid generated by a finite number of matrices in $\mathcal{M}_m(\mathbb{Z})$. This last problem was proved to be decidable [Jac78],[MS77]. To the best of our knowledge, the complexity of this problem is not known. However, for *positive linear systems* (i.e., all matrices are positive ($M_a \in \mathcal{M}_m(\mathbb{N})$)) we can easily deduce from [MS77] an algorithm to decide the finiteness of a linear system in EXPTIME. Sometimes this algorithm is useless because we consider a linear system that is clearly finite:

Proposition 1. *Transfer/Reset/Inhibitor Petri Nets [DFS98] [FS00b] and all broadcast protocols [EN98], [EFM99], [Del00] are finite linear systems.*

Proof. Just remark that the square matrices in the linear function are *Reset/Transfer/Inhibitor matrix* (*RTI-matrix*): an RTI-matrix is a matrix such that all its entries are equal to “0” expect at most one entry per column that can be equal to “1”. We see that the product of two RTI-matrices is an RTI-matrix. As there exists at most a finite number of RTI-matrices with a given size, we have proved that every linear system such that all matrices M_a are RTI-matrices, is finite. □

In practice a linear system is finite every time we have “no duplication of the variables”(the variables x_1 is said *duplicated* by f if $f(x_1, x_2) = (x_1, x_2) + v$ for every $(x_1, x_2) \in \mathbb{N}^m$).

For finite linear systems it seems natural that we can represent the σ^*-acceleration of a Presburger-definable set by a Presburger formula. However, this is not an obvious consequence because the guards used by Boigelot are only of the form $A.x \leq b$ and do not permit to represent every Presburger-definable set. In fact, remark that this class of guards are not closed by union. Also remark that in the next section, this closure by union is proved to be an important property that enables us to find the “good” words σ to accelerate. In fact, we can do it and we can even do better by computing the σ^*-acceleration relation.

Proposition 2. *The relation $s' \in f^*(s)$ is effectively equivalent to a Presburger formula if the function f is Presburger-linear and if the monoid $< M >$ is finite.*

Proof. The complete proof is available in [FL02]. Let us consider a totally-defined Presburger-linear function $f = (M, v, true)$ such that there exist two integers a and b satisfying $M^{a+b} = M^a$. For every integer $q \geq 0$, we have $f^{a+q.b}(s) = f^a(s) + q.M^a.f^b(0)$. By using this "linear equality", we deduce a Presburger formula that represents $s' \in f^*(s)$. □

We can deduce from the previous Proposition 2 the main result: for any finite linear system, all the σ^*-accelerations are computable.

Theorem 2. *For a finite linear system $\boldsymbol{L}$ and for every word $\sigma \in \Sigma^*$, the σ^*-acceleration relation is effectively Presburger-definable.*

Let us remark that the previous theorem is a powerful one because finite linear systems form a highly undecidable model. In fact, the reachability problem (is a state s in $Reach(\boldsymbol{L}, S_0)$?) is undecidable for finite linear systems of dimension $m = 3$ with $S_0 = \{s_0\}$. This can be proved by reducing the Post problem to the reachability problem with the same method than in [FB95].

Finally, remark that we can construct counter examples to the converse of the Theorem 2 just by choosing non finite linear system such that the guards of its linear functions are not satisfiable ($\phi_a = false$). However, for the class of totally defined linear systems (see Theorem 4 for another property satisfying by this class), the converse of the Theorem 2 is true.

Definition 5 (Totally defined linear systems). *A totally defined linear system is a linear system such that all its Presburger-linear functions are totally defined.*

Proposition 3. *A totally defined linear system such that for every word σ, the relation $s' \in f_\sigma^*(s)$ is Presburger-definable is finite.*

Proof. A complete proof is given in [FL02] and it is a direct consequence of [Boi98] and the Burnside theorem [MZ75], [Jac78], [MS77]. □

5 How to Find out the Good Accelerations?

For a finite linear system $\boldsymbol{L} = (\Sigma, m, f_\Sigma)$, the number of words σ with a length less than or equal to an integer $k \geq 0$ is equal to $\sum_{i=1}^{k} |\Sigma|^k$. Therefore, it seems that a brute force algorithm that computes the accelerations by all these words will take too much time. However, for finite linear systems this exponential number of words can be put together such that we have only to consider a polynomial number of accelerations ! Let us show why.

Consider two Presburger-linear functions $f_1 = (M_1, v_1, \phi_1)$ and $f_2 = (M_2, v_2, \phi_2)$ such that $\bar{f_1} = \bar{f_2}$ (recall that it is equivalent to $M_1 = M_2$ and $v_1 = v_2$) and let us define the Presburger-linear function $(M, v, \phi_1 \vee \phi_2)$ where $M = M_1 = M_2$ and $v = v_1 = v_2$ written $f_1 \vee f_2$. We then have $s' \in (f_1 \vee f_2)^*(s)$ if and only if $s' \in (f_1, f_2)^*(s)$ where $s' \in (f_1, f_2)^*(s)$ is defined by $s' \in f_1^* \circ f_2^* \circ \cdots \circ f_1^* \circ f_2^*(s)$.

Therefore, the good measure for counting the number of σ^*-accelerations with a length less than or equal to an integer $k \geq 0$ we have to consider to be exhaustive is not the number of words in $\Sigma^{\leq k}$, but the cardinal of the set $\{\bar{f}_\sigma; \sigma \in \Sigma^{\leq k}\}$ called *the number of σ^*-accelerations of length less than or equal to k.*

Theorem 3. *The number of σ^*-accelerations of length less than or equal to an integer k for a finite linear system is polynomial in the variable k.*

Proof. Consider the finite set $V_{\boldsymbol{L}} = \{M.v_a; M \in \mathcal{M}_{\boldsymbol{L}}; a \in \Sigma\}$ and a word $\sigma = a_1 \dots a_k$. We have $\bar{f}_\sigma(s) = M_\sigma.s + \sum_{i=1}^{k} M_{a_1} \cdots M_{a_{i-1}}.v_{a_i}$. Therefore $f_\sigma(s)$ is equal to $M_\sigma.s$ plus the sum of k elements in $V_{\boldsymbol{L}}$ (not necessary different). So, there exists a finite sequence of non negative integers $(\alpha_v)_{v \in V_{\boldsymbol{L}}}$ such that $\bar{f}_\sigma(s) = M_\sigma.s + \sum_{v \in V_{\boldsymbol{L}}} \alpha_v.v$ and $\sum_{v \in V_{\boldsymbol{L}}} \alpha_v = k$. As the number of sequences of non negative integers $(\alpha_v)_{v \in V}$ such that $\sum_{v \in V_{\boldsymbol{L}}} \alpha_v \leq k$ is bounded by $k^{|V_{\boldsymbol{L}}|}$, the number of σ^*-accelerations of length less than or equal to an integer k for a linear system $\boldsymbol{L} = (\Sigma, m, f_\Sigma)$ is bounded by $|\mathcal{M}_{\boldsymbol{L}}|.k^{|V_{\boldsymbol{L}}|}$. □

So, we have proved that for finite linear systems, we can increase the length of the σ^*-accelerations we consider without exponential grow up. Now the question is “Can we bound the length of the σ^*-accelerations we have to consider ?”. The exact problem is the following one: being given a linear system such that its reachability set can be computed by σ^*-accelerations, can we bound the length of the used words σ ? We have only succeeded in answering this question for the class of totally defined finite linear systems.

Theorem 4. *Consider a totally defined finite linear system $\boldsymbol{L}$, the reachability relation is Presburger-definable. Moreover, we can compute t words σ_1, ..., σ_t with length less than or equal to $|\mathcal{M}_{\boldsymbol{L}}|$ such that the reachability relation is equal to $f^*_{\sigma_1} \circ \cdots \circ f^*_{\sigma_t}$.*

Proof. In the complete proof available in [FL02], we consider the automaton $A_{\boldsymbol{L}} = (Q, \Sigma, \delta_{\boldsymbol{L}})$ defined by a set of states $Q = \mathcal{M}_{\boldsymbol{L}}$ and a relation $\delta_{\boldsymbol{L}}$ such that $\delta_{\boldsymbol{L}}(a, M) = M.M_a$ for every $a \in \Sigma$ and for every $M \in \mathcal{M}_{\boldsymbol{L}}$. We prove that the linear function associated to a sequence of transitions σ corresponds to the label of a path in the automaton $A_{\boldsymbol{L}}$. By commutativity, we then prove that we have just to consider the accelerations of the elementary cycles of the automaton $A_{\boldsymbol{L}}$ to obtain the reachability relation. □

This class of totally defined finite linear systems is therefore a really important one because it permits us to obtain for a finite linear system an over approximation of its reachability relation by considering the reachability relation of its naturally associated totally defined finite linear system.

6 Fast Acceleration of Symbolic Transition Systems

`FAST` is a tool that takes as input a finite linear system $\boldsymbol{L}$ and a set of initial states S_0 and computes recursively the set $f^*_{\sigma_n} \circ \cdots \circ f^*_{\sigma_1}(S_0)$ where the σ_i are chosen in $\Sigma^{\leq k}$. Some technical heuristics are implemented in FAST to increase the integer k when the computation seems to diverge.

With these heuristics, the reachability sets of 8 cache coherence protocols (Broadcast protocols) is computed by FAST in less than 10 seconds on an Intel Pentium 800 and the value of k took the value 1. The reachability set of the SWIMMING POOL protocol [FO97b] is also obtained by FAST after 3 minutes and the integer k took the value 4. Moreover, the automata which represent these reachability sets have a very small number of states. The following table gives precisely the obtained results. The specification of all these protocols can be found on our home page `http://www.lsv.ens-cachan.fr/~leroux/fast/`.

Protocols	$\|\Sigma\|$	m	$\|A_{\phi_0}\|$	$\|A_{Reach}\|$	k
BERKELEY	3	4	4	7	1
DRAGON	8	5	4	7	1
FIREFLY	8	4	4	7	1
FUTURBUS	10	9	4	12	1
ILLINOIS	6	4	4	7	1
MESI	4	4	4	5	1
MOESI	4	5	4	7	1
SYNAPSE	3	3	4	6	1
SWIMMING POOL	6	9	6	29	4
PRODUCER-CONSUMER	3	5	3	6	1

Conclusion and Perspectives

We have proved that we can compute accelerations of any sequence of actions in a finite linear system. Moreover, we have given a method to find out the "good accelerations" by reducing the number of different accelerations we have to consider. These methods are implemented in the tool FAST. Even if we can find out some finite linear systems that have a Presburger-definable set of reachable states but such that FAST does not stop (an example is given in [FL02]), this tool enables to compute the set of reachable states of a great number of protocols. Finally, by considering the totally defined linear system associated to a finite linear system, we have proved that we can compute an over approximation of the set of reachable states.

Finally let us make a remark about the commutativity of the functions and the reduction of the number of "good accelerations". We have proved that if we have two Presburger-linear functions f and g such that $\bar{f} \circ \bar{g} = \bar{g} \circ \bar{f}$ then we can replace the set $\{f, g, f \circ g, g \circ f\}$ by $\{f, g, f \circ g \vee g \circ f\}$. However, sometimes we also have $f \circ g = g \circ f$. In this case, we can replace $\{f, g, f \circ g, g \circ f\}$ by $\{f, g\}$. Commutativity enables us to reduce the number of different accelerations. However, it seems that commutativity has not been used as best as possible.

References

[AAB99] P. Abdulla, A. Annichini, and A. Bouajjani. Symbolic verification of lossy channel systems: Application to the bounded retransmission protocol. In *(TACAS'99)*, volume 1579 of *LNCS*, pages 208–222. Springer-Verlag, 1999.

[AAB00] A. Annichini, E. Asarin, and A. Bouajjani. Symbolic techniques for parametric reasoning about counter and clock systems. In *(CAV'00)*, volume 1855 of *LNCS*, pages 419–434. Springer-Verlag, 2000.

[AJ96] P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Information and Computation*, 127(2):91–101, june 1996.

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model checking. In *(CONCUR'97)*, volume 1243 of *LNCS*, pages 135–150. Springer-Verlag, June 1997.

[BF99] B. Bérard and L. Fribourg. Reachability analysis of (timed) Petri nets using real arithmetic. In *(CONCUR'99)*, volume 1664 of *LNCS*, pages 178–193. Springer-Verlag, 1999.

[BF00] J.-P. Bodeveix and M. Filali. Fmona: a tool for expressing validation techniques over infinite state systems. In *(TACAS'00)*, volume 1785 of *LNCS*, pages 204–219. Springer-Verlag, march 2000.

[BGP97] T. Bultan, R. Gerber, and W. Pugh. Symbolic model checking of infinite state systems using presburger arithmetic. In *(CAV'97)*, volume 1254 of *LNCS*, pages 400–411. Springer-Verlag, 1997.

[BGWW97] B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *(SAS'97)*, volume 1302 of *LNCS*, pages 172–186. Springer-Verlag, 1997.

[BH99] A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theoretical Computer Science*, 221(1–2):211–250, June 1999.

[BJNT00] A. Bouajjani, B. Jonsson, M. Nilsson, and T. Touili. Regular model checking. In *(CAV'00)*, volume 1855 of *LNCS*, pages 403–418. Springer-Verlag, July 2000.

[Boi98] B. Boigelot. *Symbolic Methods for Exploring Infinite State Spaces*. PhD thesis, Université de Liège, 1998.

[BW94] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *(CAV'94)*, volume 818 of *LNCS*, pages 55–67. Springer-Verlag, 1994.

[CJ98] H. Comon and Y. Jurski. Multiple counters automata, safety analysis and presburger arithmetic. In *(CAV'98)*, volume 1427 of *LNCS*, pages 268–279. Springer-Verlag, 1998.

[Del00] Giorgio Delzann. Automatic verification of parameterized cache coherence protocols (extended and revised version). In *(CAV'00)*, volume 1855 of *LNCS*, pages 53–68. Springer-Verlag, 2000.

[DFS98] C. Dufourd, A. Finkel, and P. Schnoebelen. Reset nets between decidability and undecidability. In *(ICALP'98)*, volume 1443 of *LNCS*, pages 103–115. Springer-Verlag, July 1998.

[EFM99] J. Esparza, A. Finkel, and R. Mayr. On the verification of broadcast protocols. In *(LICS'99)*, pages 352–359. IEEE Computer Society, July 1999.

[EN98] A. Emerson and K. Namjoshi. On model checking for non-deterministic infinite-state systems. In *(LICS'98)*, pages 70–80, 1998.

[FAS] FAST. http://www.lsv.ens-cachan.fr/ leroux/fast/.

[FB95] R. Floyd and R. W Beigel. *Le langage des machines : introduction a la calculabilite et aux langages formels*, chapter 7, page 433. (ITP 1995), 1995.

[FL02] A. Finkel and J. Leroux. How to compose Presburger-accelerations: Applications to broadcast protocols. Technical report, Laboratoire Spécification et Vérification, CNRS UMR 8643 & ENS de Cachan, 2002.

[FO97a] L. Fribourg and H. Olsén. A decompositional approach for computing least fixed-points of Datalog programs with Z-counters. *Constraints*, 2(3/4):305–335, 1997.

[FO97b] L. Fribourg and H. Olsén. Proving safety properties of infinite state systems by compilation into Presburger arithmetic. In *(CONCUR'97)*, volume 1243 of *LNCS*, pages 213–227. Springer-Verlag, 1997.

[FPS00] A. Finkel, S. Purushothaman Iyer, and G. Sutre. Well-abstracted transition systems. In *(CONCUR'2000)*, volume 1877 of *LNCS*, pages 566–580. Springer-Verlag, 2000.

[FS00a] A. Finkel and G. Sutre. An algorithm constructing the semilinear post* for 2-dim reset/transfer vass. In *(MFCS'2000)*, volume 1893 of *LNCS*, pages 353–362. Springer-Verlag, 2000.

[FS00b] A. Finkel and G. Sutre. Decidability of reachability problems for classes of two counters automata. In *(STACS'2000)*, volume 1770 of *LNCS*, pages 346–357. Springer-Verlag, Feb 2000.

[FWW97] A. Finkel, B. Willems, and P. Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In *(INFINITY'97)*, volume 9 of *Electronical Notes in Theoretical Computer Science.* Elsevier Science, July 1997.

[GS66] S. Ginsburg and E. H. Spanier. Semigroups, Presburger formulas and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

[Jac78] G. Jacob. La finitude des representations lineaires des semi-groupes est decidable. *J. Algebra*, 52:437–459, 1978.

[MON] MONA. http://www.brics.dk/mona/index.html.

[MS77] A. Mandel and I. Simon. On finite semigroups of matrices. *Theoretical Computer Science*, 5(2):101–111, October 1977.

[MZ75] R. McNaughton and Y. Zalcstein. The burnside theorem for semi-groups. *J. Algebra*, 34:292–299, 1975.

[PS00] A. Pnueli and E. Shahar. Liveness and acceleration in parameterized verication. In *(CAV'00)*, volume 1855 of *LNCS*, pages 328–343. Springer-Verlag, July 2000.

[Rev90] P. Z. Revesz. A closed form for Datalog queries with integer order. In *ICDT'90, Third International Conference on Database Theory*, volume 470 of *LNCS*, pages 187–201. Springer-Verlag, December 1990.

[SFRC99] G. Sutre, A. Finkel, O. Roux, and F. Cassez. Effective recognizability and model checking of reactive fiffo automata. In *(AMAST'98)*, volume 1548 of *LNCS*, pages 106–123. Springer-Verlag, 1999.

[Sut00] G. Sutre. *Abstraction et accélération de systèmes infinis.* PhD thesis, Ecole Normale Supérieure de Cachan, Laboratoire Spécification et Vérification. CNRS UMR 8643, october 2000.

[WB98] P. Wolper and B. Boigelot. Verifying systems with infinite but regular state spaces. In *(CAV'98)*, volume 1427 of *LNCS*, pages 88–97, June 1998.

State Space Reductions for Alternating Büchi Automata
Quotienting by Simulation Equivalences

Carsten Fritz and Thomas Wilke

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität, 24098 Kiel, Germany
{fritz,wilke}@ti.informatik.uni-kiel.de

Abstract. Quotienting by simulation equivalences is a well-established technique for reducing the size of nondeterministic Büchi automata. We adapt this technique to *alternating* Büchi automata. To this end we suggest two new quotients, namely minimax and semi-elective quotients, prove that they preserve the recognized languages, and show that computing them is not more difficult than computing quotients for nondeterministic Büchi automata. We explain the merits of of our quotienting procedures with respect to converting alternating Büchi automata into nondeterministic ones.

1 Introduction

Minimizing ω-automata is computationally difficult, because testing universality for nondeterministic finite automata on strings is already PSPACE-hard [GJ79][1]. Nevertheless, state-space reduction for ω-automata is an important issue in verification. Therefore, various state-space reduction heuristics have been developed, most of them for nondeterministic Büchi automata. Many of these heuristics are based on quotienting by bisimulation or simulation equivalences, that is, a given automaton is reduced in size by first computing an equivalence relation that identifies structurally "similar" states and then building a quotient with respect to this equivalence relation. In this paper, we show how this technique can be used to reduce the size of alternating Büchi automata.

Motivation. In the automata-theoretic approach to trace-based model checking [VW86] [VW94], a linear-time temporal formula (spec) is checked against a given transition system in three steps. First, the negation of the spec is translated into an equivalent nondeterministic Büchi automaton. Second, a product of the automaton and the given system is computed. Third, an emptiness test is performed for the product automaton. This solves the problem since the product automaton will accept exactly the error traces of the system with respect to the given spec. Recent work [GO01] suggests to split up the first step: the spec is translated into a (weak) alternating Büchi automaton, then, this automaton is simplified, and finally, the simplified automaton is translated into a nondeterministic Büchi automaton. Clearly, the size of the resulting nondeterministic Büchi

[1] This also implies that approximation of a minimum-size ω-automaton within a constant factor is impossible in polynomial time unless P=PSPACE.

automaton can be reduced by appropriate quotienting. However, an early quotienting of the alternating automaton may speed up the whole process.

Model-checking techniques that do not follow an automata-theoretic paradigm often interpret a given spec as a fixed-point formula and evaluate this formula on the transition system in question. This applies to various specification formalisms, for instance, CTL [McM93], and when symbolic techniques are used. Sometimes, a fixed-point logic can even be used directly for specification [LN01]. Since almost all specification formalisms and especially fixed-point logics can easily be translated into alternating automata and, conversely, alternating automata can be viewed as fixed-point formulas, see, e. g., [EJ88,MSS88,EJ91,Var94,KVW00], the following procedure suggests itself: translate the formula into an alternating automaton, reduce the automaton, interpret the reduced automaton as a fixed-point formula, and evaluate this formula on the transition system. That is, reducing automata is used for minimizing formulas. In the second step of the above process, quotienting by simulation equivalences might prove to be useful. As a start, one may want to study quotienting for alternating Büchi automata.

New Results. We first adapt direct, delayed, fair and ordinary simulation to alternating Büchi automata; this combines the definitions of [ESW01] for nondeterministic Büchi automata with the definition of [AHKV98] for alternating transition systems (Sect. 3). We then observe that because of alternation there is no simple way to define appropriate quotients with respect to any of the simulation equivalences. We suggest two new quotients, minimax and semi-elective quotients (Subsections 4.1 and 4.2, respectively), and show that these quotients preserve the languages recognized and can thus be used for state-space reduction with alternating Büchi automata. The proofs are quite complicated and cannot be given in this extended abstract; our main technical tool is a notion of composition of simulation strategies (see [FW02, Sect. 4]). We discuss how quotienting of alternating Büchi automata can be used to speed up the conversion of alternating Büchi automata to nondeterministic ones (Sect. 5). Finally, we explain why the efficient algorithms from [ESW01] can be used to compute our simulation relations and quotients for alternating Büchi automata and that computing these relations and quotients is especially easy for weak alternating Büchi automata (Sect. 6).

This extended abstract gives a semi-formal treatment of the material; for the exact details, the reader is referred to the full version, which is on-line, see [FW02].

Related Work. Henzinger, Kupferman, and Rajamani [HKR97,HR00] introduce fair bisimulation and simulation relations and describe how they can be computed efficiently. Somenzi and Bloem [SB00] and Etessami and Holzmann [EH00] use direct simulation to reduce the size of nondeterministic Büchi automata in the context of checking linear-time temporal properties and present efficient algorithms for computing direct simulation. Etessami, Schuller, and Wilke [ESW01] improve on [HKR97,HR00] and introduce delayed simulation to obtain better reductions; they make use of Jurdziński's algorithm [Jur00], which solves parity games. Gurumurthy, Bloem, and Somenzi [GBS02] build on this; Etessami [Ete02] follows different directions. Fast algorithms for computing fair simulation using games were also presented by Bustan and Grumberg [BG00]. Alur, Henzinger, Kupferman, and Vardi [AHKV98] study ordinary simulation for alternating transition systems. Gastin and Oddoux [GO01] use very weak alternating Büchi automata for translating linear-time temporal formulas into Büchi automata; they suggest some simplification rules for alternating Büchi automata, see above. Generating Büchi automata from linear-time temporal formulas is also dealt with in [GPVW95,DGV99].

There is more work on simulation in general, see, e. g., [Mil71,HHK95], and on using simulation for testing language inclusion in verification, see, e. g., [DHW91].

2 Notation and Basic Definitions

The set of natural numbers is denoted ω. Words over some alphabet Σ are viewed as functions from an initial segment of ω or ω itself to Σ, so when w is a word, then $w(i)$ denotes the letter at its ith position, where the first letter is in position 0.

For the purpose of this paper, an alternating Büchi automaton is a tuple

$$A = (Q, \Sigma, q_I, \Delta, E, U, F) \ , \tag{1}$$

where Q is a finite *set of states,* Σ a finite *alphabet,* $q_I \in Q$ an *initial state,* $\Delta \subseteq Q \times \Sigma \times Q$ a *transition relation,* $\{E, U\}$ a partition of Q in *existential* and *universal states,* where $E = \emptyset$ and $U = \emptyset$ are allowed, and $F \subseteq Q$ a set of *accepting states.* For notational simplicity, we will often write $\Delta(q, a)$ for $\{q' \mid (q, a, q') \in \Delta\}$.

Acceptance of alternating Büchi automata is best defined via games. For an alternating Büchi automaton A as above and an ω-word $w \in \Sigma^\omega$, the *word game* $G(A, w)$ is the Büchi game

$$G = (P, P_0, P_1, p_I, Z, F') \tag{2}$$

where $P = Q \times \omega$ is the set of positions, $P_0 = U \times \omega$ is the set of positions of Player 0, $P_1 = E \times \omega$ is the set of positions of Player 1, $p_I = (q_I, 0)$ is the initial position, $Z = \{((s, i), (s', i+1)) \mid (s, w(i), s') \in \Delta\}$ is the set of moves, and $F' = F \times \omega$ is the set of accepting positions. A *play* of the game is a finite or infinite sequence of positions $\pi = (p_i)_{i<n}$ (with $n \in \omega \cup \{\omega\}$) such that $p_0 = p_I$, $(p_i, p_{i+1}) \in Z$ for every $i+1 < n$, and $n < \omega$ only if there is no $p \in P$ such that $(p_{n-1}, p) \in Z$. A play π is a win for Player 1 if either infinitely many positions of π belong to F', or $n < \omega$ and $p_{n-1} \in P_0$; else π is a win for Player 0. Notice that the winning condition is phrased for Player 1, as opposed to conventions used in other papers.

Following [GH82], Player 0 will be called *Pathfinder* while Player 1 will be called *Automaton*. Acceptance is now defined as follows. The word w is *accepted* by the automaton A if Automaton wins the game $G(A, w)$, i. e., if Automaton has a winning strategy for $G(A, w)$. The language *recognized* by A is

$$L(A) = \{w \in \Sigma^\omega \mid \text{Automaton wins the game } G(A, w)\} \ . \tag{3}$$

For $q \in Q$, we will write $A(q)$ for the *translation* of A to q, which is defined to be the same automaton but with initial state q, i. e., $A(q) = (Q, \Sigma, q, \Delta, E, U, F)$.

In figures, existential states are shown as diamonds and universal states as squares; accepting states have double lines, see, e. g., Fig. 1.

3 Simulation Relations

In this section, we define three types of simulation relations for alternating Büchi automata, namely direct, delayed, and fair simulation, which are all based on the same simple game, only the winning condition varies; we follow the approach of [ESW01]. We also state basic properties of the three types of simulation relations, in particular, that simulation implies language containment.

Let $A^0 = (Q^0, \Sigma, p_I, \Delta^0, E^0, U^0, F^0)$ and $A^1 = (Q^1, \Sigma, q_I, \Delta^1, E^1, U^1, F^1)$ be alternating Büchi automata. The *basic simulation game* $G(A^0, A^1)$ is played by two players, *Spoiler* and *Duplicator,* who play the game in rounds. At the beginning of each round, a pair (p, q) of states $p \in Q^0$ and $q \in Q^1$ is given, and the players play as follows.

1. Spoiler chooses a letter $a \in \Sigma$.

2. Spoiler and Duplicator play as follows, depending on the modes of p and q.

— If $(p, q) \in E^0 \times E^1$, then Spoiler chooses a transition $(p, a, p') \in \Delta^0$ and after that Duplicator chooses a transition $(q, a, q') \in \Delta^1$.

— If $(p, q) \in U^0 \times U^1$, then Spoiler chooses a transition $(q, a, q') \in \Delta^1$ and after that Duplicator chooses a transition $(p, a, p') \in \Delta^0$.

— If $(p, q) \in E^0 \times U^1$, then Spoiler chooses transitions $(p, a, p') \in \Delta^0$ and $(q, a, q') \in \Delta^1$.

— If $(p, q) \in U^0 \times E^1$, then Duplicator chooses transitions $(p, a, p') \in \Delta^0$ and $(q, a, q') \in \Delta^1$.

3. The starting pair for the next round is (p', q').

The first round begins with the pair (p_I, q_I). If, at any point during the course of the game, a player cannot proceed any more, he or she looses (early). When the players proceed as above and no player looses early, they construct an infinite sequence $(p_0, q_0), (p_1, q_1), \ldots$ of pairs of states (with $p_0 = p_I$ and $q_0 = q_I$), and this sequence determines the winner, depending on the type of simulation relation we are interested in:

Direct simulation (di): Duplicator wins if for every i with $p_i \in F^0$ we have $q_i \in F^1$.

Delayed simulation (de): Duplicator wins if for every i with $p_i \in F^0$ there exists $j \geq i$ such that $q_j \in F^1$.

Fair simulation (f): Duplicator wins if there are only finitely many i with $p_i \in F^0$, or infinitely many j with $q_j \in F^1$.

In all other cases, Spoiler wins. This completes the description of the games.

When the basic game $G(A^0, A^1)$ is provided with one of the above winning conditions we write $\Game A^0 A^1 di$, $\Game A^0 A^1 de$, or $\Game A^0 A^1 f$ for this game. For $x \in \{di, de, f\}$, we define a relation $\leq_x$ on alternating Büchi automata. We write $A \leq_x B$ when Duplicator has a winning strategy in $\Game ABx$ and say that B *x-simulates* A. For states p of A, q of B, we write $p \leq_x q$ to indicate that $B(q)$ x-simulates $A(p)$. We write $\Game pqx$ instead of $\Game A(p)B(q)x$ when A and B are obvious from the context.

As an example, consider the automaton A over the alphabet $\Sigma = \{a, b\}$ given in Fig. 1. We argue that Duplicator wins $\Game 01x$ for $x \in \{de, di, f\}$. If Spoiler does not want to loose early, he has to choose letter b and transition $(1, b, 2)$ in the first round. Duplicator can reply by choosing transition $(0, b, 2)$, and the second round starts in position $(2, 2)$. Regardless of what Spoiler chooses in the second round, a and $(2, a, 2)$ or, alternatively, b and $(2, b, 2)$, Duplicator can reply such that after the second round, the game is in position $(2, 2)$ again. The same holds true for any further round. Consequently, $0 \leq_x 1$ holds. Observe that $1 \leq_x 0$ is true for $x \in \{de, f\}$, but not for $x = di$.

For simplicity, we henceforth assume without loss of generality that all alternating Büchi automata are complete, which means that for every $q \in Q$, $a \in \Sigma$, there is a $q' \in Q$ such that $(q, a, q') \in \Delta$. Only in examples, we still allow incomplete automata in order to keep the examples small.

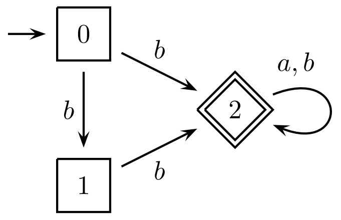


Fig. 1. Alternating Büchi automaton

It is easy to see that the three simulation relations are increasingly coarser:

Lemma 1. *The following relations hold between the three types of simulation relations:* $\leq_{di} \subsetneq \leq_{de} \subsetneq \leq_f$.

Unlike in the case of nondeterministic automata, it is not trivial to see that the three types of simulation relations are preorders, which the notation suggests. Using the strategy-composition method described in [FW02, Sect. 4], we are able to show this is, in fact, true:

Proposition 2 (simulations are preorders). *The simulation relations* $\leq_{di}$*,* $\leq_{de}$*, and* $\leq_f$ *are preorders, i. e., they are reflexive and transitive.*

The corresponding *simulation equivalence relations* are denoted by $\equiv_{de}$, $\equiv_{di}$, and $\equiv_f$, respectively, that is, $A \equiv_x B$ if $A \leq_x B$ and $B \leq_x A$. An analogue notation is used for states, that is, $p \equiv_x q$ if $A(p) \leq_x B(q)$ and $B(q) \leq_x A(p)$. The respective equivalence classes are denoted $[q]_x$.

We prove that all three types of simulations imply language containment and can thus be used for checking language containment:

Theorem 3 (simulation implies language containment). *Let* $x \in \{di, de, f\}$ *and let* A^0 *and* A^1 *be alternating Büchi automata.*

If $A^0 \leq_x A^1$*, then* $L(A^0) \subseteq L(A^1)$.

4 Simulation Quotients

In this section, we study how quotienting should be performed with respect to the three types of simulation relations. On the one hand, we will explain why the naive quotient construction—the construction studied in [ESW01] for nondeterministic Büchi automata and proved to work with such automata—does not work with alternating Büchi automata. On the other hand, we present two new quotient constructions, namely minimax and semi-elective quotients, which do work.

In general, when $\equiv$ is an equivalence relation on the state space of an alternating Büchi automaton A, we call an alternating Büchi automaton a *quotient of* A with respect to $\equiv$ if it is of the form

$$(Q/{\equiv}, \Sigma, [q_I]_{\equiv}, \Delta', E', U', F/{\equiv}) \tag{4}$$

and satisfies the following natural constraints for all $q, q' \in Q$, $a \in \Sigma$:

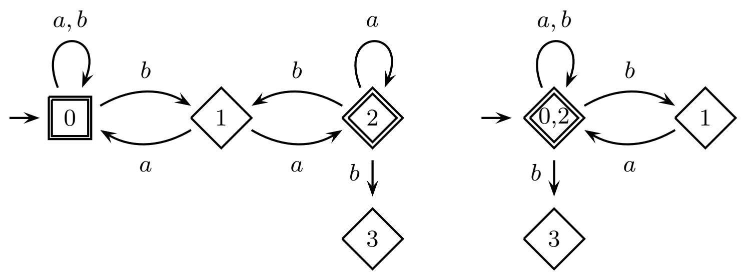


Fig. 2. Naive quotients don't work

(1) if $([q]_\equiv, a, [q']_\equiv) \in \Delta'$, then there exist states $\hat{q}$ and $\hat{q}'$ such that $\hat{q} \equiv q$, $\hat{q}' \equiv q'$ and $(\hat{q}, a, \hat{q}') \in \Delta$,

(2) if $[q]_\equiv \subseteq E$, then $[q]_\equiv \in E'$, and

(3) if $[q]_\equiv \subseteq U$, then $[q]_\equiv \in U'$.

Note that these conditions are minimal requirements so that a quotient really reflects the structure of A and is not just any automaton on the equivalence classes of $\equiv$. A quotient is a *naive quotient* if the converse holds true in (1), that is, if transitions are representative-wise.

In [ESW01], naive quotients of nondeterministic Büchi automata were proved to preserve the recognized language. This is no longer true for alternating Büchi automata, as problems arise for mixed simulation equivalence classes, i. e., classes containing both existential and universal states. In the naive quotienting, these states can be declared either existential or universal, but regardless of how they are declared the resulting naive quotients may *not* be equivalent to the original automaton. Consider, for instance, Fig. 2, showing an alternating Büchi automaton A over $\Sigma = \{a, b\}$ on the left, where $0 \equiv_x 2$ for $x \in \{de, di, f\}$, and one of the two possible naive x-quotients on the right; the other is obtained by declaring the state $\{0, 2\}$ universal. Now observe that the automaton on the left-hand side does not accept b^ω but $(ba)^\omega$, while the quotient on the right-hand side accepts all words and the other quotient does not accept $(ba)^\omega$. So neither one of the quotients is equivalent to the original automaton.

4.1 Direct Simulation and Minimax Quotients

We overcome the problems with direct simulation quotienting by using a more sophisticated transition relation for the quotient automaton which exploits the simple structure of direct simulation games.

Let q be a state of an alternating Büchi automaton A and $a \in \Sigma$. A state $q' \in \Delta(q, a)$ is an *x-maximal a-successor of q* if $q'' \leq_x q'$ holds for every $q'' \in \Delta(q, a)$ with $q' \leq_x q''$. We define

$$\max^x_a(q) = \{q' \in \Delta(q, a) \mid q' \text{ is an } x\text{-maximal } a\text{-successor of } q\} \quad . \tag{5}$$

Symmetrically, *x-minimal a-successors of q* and $\min^x_a(q)$ are defined.

An *x-minimax quotient* of an alternating Büchi automaton A as in (1) is a quotient where the transition relation is given by

$$\begin{aligned}\Delta_x^m = \{([p]_x, a, [q]_x) \mid a \in \Sigma, p \in E, q \in \max_a^x(p)\} \\ \cup \{([p]_x, a, [q]_x) \mid a \in \Sigma, p \in U, q \in \min_a^x(p)\} \ . \end{aligned} \quad (6)$$

In a minimax quotient it does not matter how the mixed states are declared. This is not surprising, as we prove that a mixed class is deterministic in any minimax quotient: for every letter there is at most one outgoing edge labelled with that letter.

The main result about minimax quotients we prove is:

Theorem 4 (minimax quotients). *Let A be an alternating Büchi automaton as in (1) and B^m any di-minimax quotient of A.*

1) For all $p, q \in Q$ such that $p \leq_{di} q$, $A(q)$ di-simulates $B^m([p]_{di})$ and $B^m([q]_{di})$ di-simulates $A(p)$, that is, $[p] \leq_{di} q$ and $p \leq_{di} [q]_{di}$.

2) A and B^m di-simulate each other, that is, $A \equiv_{di} B^m$.

3) A and B^m are equivalent, that is, $L(A) = L(B^m)$.

4.2 Delayed Simulation and Semi-elective Quotients

For delayed simulation, neither naive quotients nor minimax quotients work. That naive quotients do not work follows from our previous example, see Fig. 2. To see that minimax quotients do not work, consider the automaton in Fig. 3. It is easy to see that $0 \geq_{de} 1$ but not $0 \equiv_{de} 1$, i.e., $\max_a^{de}(0) = \{0\}$. Therefore, the de-minimax quotient of the automaton has no transition from $[0]_{de}$ to $[1]_{de}$ and thus recognizes the empty language, which is not true for the automaton itself.

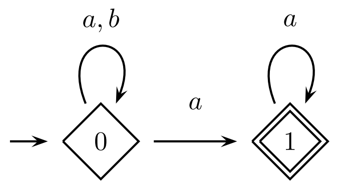


Fig. 3. De-minimax quotients don't work

To overcome the problem, we define so-called semi-elective quotients. A *semi-elective quotient* of an alternating Büchi automaton A is the quotient where the transition relation is given by

$$\begin{aligned}\Delta_x^s = \{([p]_x, a, [q]_x) \mid (p, a, q) \in \Delta, p \in E\} \\ \cup \{([p]_x, a, [q]_x) \mid a \in \Sigma, [p]_x \subseteq U, q \in \min_a^x(p)\}, \end{aligned} \quad (7)$$

and every mixed class is existential. That is, purely universal classes are treated as with minimax quotienting while purely existential and mixed classes are existential states and have representative-wise transitions. Given an alternating Büchi automaton A and $x \in \{de, di, f\}$, we write A_x^s for the respective semi-elective quotient.

Some possible optimizations of this construction are discussed in [FW02, Subsect. 7.4].

The main result about semi-elective quotients we prove is:

Theorem 5 (semi-elective quotients). *Let A be an alternating Büchi automaton as in (1) and $x \in \{de, di\}$.*

1) For all $p, q \in Q$ such that $p \leq_x q$, $A(q)$ x-simulates $A^s_x([p]_x)$ and $A^s_x([q]_x)$ x-simulates $A(p)$, that is, $[p] \leq_x q$ and $p \leq_x [q]_x$.

2) A and A^s_x x-simulate each other, that is, $A \equiv_x A^s_x$.

3) A and A^s_x are equivalent, that is, $L(A) = L(A^s_x)$.

The reader may want to verify that the automaton depicted in Fig. 1 shows that for universal classes in de-semi-elective quotients it is necessary that only de-min-successors are taken.

As an example for the construction of a semi-elective quotient automaton modulo delayed simulation, consider Fig. 4. For the automaton A on the left, we have $2 <_{de} 1 \equiv_{de} 5 <_{de} 0 \equiv_{de} 3 <_{de} 4$. Thus there are four states in the quotient automaton A^s_{de} on the right. Since $\min^{de}_b(1) = \{2\}$, the edge $([1]_{de}, b, [1]_{de})$ is not in Δ^s_{de}; since $\min^{de}_a(0) = \min^{de}_b(0) = \{1\}$, there is no edge $([0]_{de}, c, [3]_{de})$ in Δ^s_{de} with $c \in \{a, b\}$. And since $\min^{de}_a(3) = \min^{de}_b(3) = \{1\}$, there is no edge $([3]_{de}, c, [4]_{de})$ in Δ^s_{de} with $c \in \{a, b\}$. Consequently, the state $[4]_{de}$ is not reachable in A^s_{de} and should be removed in a successive optimization step of the quotient automaton.

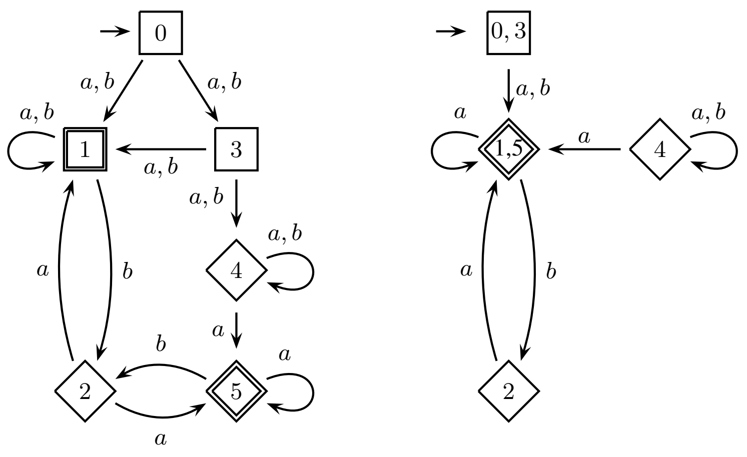


Fig. 4. Automaton and de-semi-elective quotient

5 Converting Alternating Büchi Automata to Nondeterministic Büchi Automata

Given an alternating Büchi automaton A, the standard approach for constructing an equivalent nondeterministic (i. e., non-alternating) Büchi automaton is the construction of Miyano and Hayashi [MH84]. Their construction is a modified power set construction where the states are *pairs* of subsets of the state set of A. We will call the automaton resulting from the Miyano–Hayashi construction the MH-automaton and denote it by A_{nd}. Note that A_{nd} is an exponential size automaton (and that this is necessarily so in the worst case).

In order to construct a small nondeterministic Büchi automaton from an alternating Büchi automaton, our simulation quotienting can be applied to the alternating automaton prior to the Miyano–Hayashi construction and a subsequent simulation quotienting. (The subsequent simulation quotienting should still be done.) Traditionally, simulation quotienting is only applied to the MH-automaton.

Figure 5 shows these two possible ways.

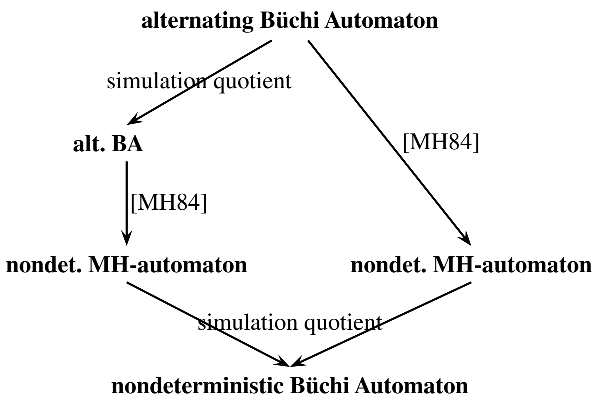


Fig. 5. Two ways from alternating BA to nondet. BA

First applying simulation quotienting to the alternating automaton is relatively cheap as compared to simulation quotienting the MH-automaton (cf. Sect. 6), since the MH-construction incurs an exponential growth (see above). For this reason, a state space reduction of the alternating automaton often results in a substantial reduction of the size of the MH-automaton. Aside from these savings in the state space, a smaller intermediate MH-automaton speeds up the subsequent simulation quotienting.

The following lemma states that the Miyano-Hayashi-construction preserves the simulation preorder.

Lemma 6. *Let A^0 and A^1 be alternating Büchi automata and $x \in \{di, de, f\}$. If $A^0 \leq_x A^1$, then $A^0_{nd} \leq_x A^1_{nd}$.*

Lemma 6 obviously implies the following corollary.

Corollary 7. *For every ABA A and $x \in \{di, de\}$, $((A^x)_{nd})^x \equiv_x (A_{nd})^x$ holds.*

Proof. We have $A^x \equiv_x A$, hence, by Lemma 6, $(A^x)_{nd} \equiv_x A_{nd}$, and $((A^x)_{nd})^x \equiv_x (A_{nd})^x$ follows immediately. □

That is, the original alternating automaton, the intermediate automata of Fig. 5 and the resulting nondeterministic Büchi automaton are all simulation equivalent.

But note that the simulation quotients of simulation equivalent (alternating or nondeterministic) automata need not be isomorphic: Some additional optimizations of the quotient construction (for example, using pseudo-accepting states as described in [FW02, Subsect. 7.4]) work to a lesser extent for A_{nd} than for A. So, when using these optimizations, the quotient resulting from taking the left-hand way in Fig. 5 is, for certain instances, smaller than the result of the right-hand way. (Without additional optimizations, the left-hand quotient will not be smaller than the right-hand quotient.)

Anyway, taking the left-hand way always results in an automaton which is no larger than the automaton arrived at by taking the right-hand way:

Proposition 8. *For every ABA A and $x \in \{di, de\}$, $((A^x)_{nd})^x$ has at most as many states as $(A_{nd})^x$.*

As an example for the possible state space reduction by quotienting of alternating automata, consider Fig. 6 showing A_3, an alternating automaton with three sub-automata B^1, B^2, B^3 (with initial states q_0^1, q_0^2, q_0^3) of sizes $2, 3, 4$, respectively. The automaton A_3 can be generalized to an automaton A_n with n sub-automata of sizes $2, \ldots, n+1$ in an obvious manner.

Then, A_n has $\theta(n^2)$ states while $(A_n)_{nd}$ has $2^{\theta(n \log n)}$ states, i. e., the size of $(A_n)_{nd}$ is in $2^{O(\sqrt{m} \log m)}$ where m is the size of A_n. But A_n^{de} only has three states since $q_0^1 \leq_{de} q_0^k$ for $k > 1$, i. e., the transitions from q_I^n to $q_0^2, \ldots, q_0^n$ are removed by de-semi-elective quotienting (and $(A_n^{de})_{nd}$ also has only three states).

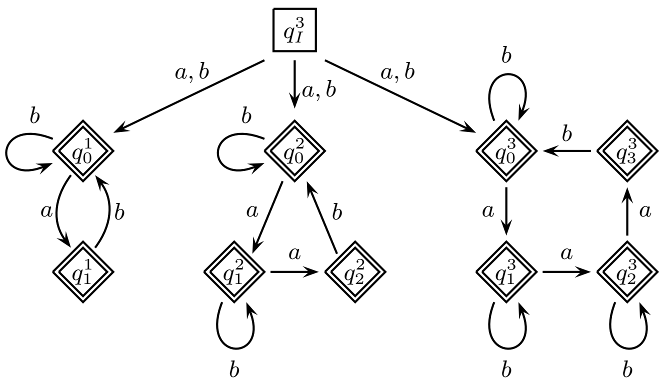


Fig. 6. A_3

6 Efficient Algorithms

Efficient algorithms for computing simulation relations and simulation quotients of nondeterministic Büchi automata are given in [ESW01], the main idea being a reduction to parity games. Using similar reductions, we obtain the same complexity bounds:

Theorem 9 (computing simulation relations). *In the following, n stands for the number of states and m for the number of transitions of an alternating Büchi automaton.*

Direct simulation relations can be computed in time $O(nm)$. Delayed and fair simulation relations can be computed in time $O(n^3 m)$ and space $O(nm)$. For weak alternating automata, direct as well as delayed and fair simulation can be computed in time $O(nm)$.

We mention that computing the respective quotients, regardless of which type they are, can be done within the same complexity bounds.

7 Conclusion

We have presented new quotients for alternating Büchi automata which preserve the recognized languages, can be computed in polynomial time, and yield considerable reductions in size (just as for nondeterministic Büchi automata). They can be used to speed up explicit-state as well as symbolic model-checking. For practical applications, these reductions should be combined with the other known state-space reduction techniques.

References

[AHKV98] Rajeev Alur, Thomas A. Henzinger, Orna Kupferman, and Moshe Y. Vardi. Alternating refinement relations. In D. Sangiorgi and R. de Simone, editors, *CONCUR 1998*, vol. 1466 of *LNCS*, pp. 163–178, 1998.

[BG00] Doran Bustan and Orna Grumberg. Checking for fair simulation in models with Büchi fairness constraints, Dec. 2000. Tech. Rep. TR-CS-2000-13, Technion.

[DGV99] Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for linear time temporal logic. In N. Halbwachs and D. Peled, editors, *CAV 1999*, vol. 1633 of *LNCS*, pp. 249–260, 1999.

[DHW91] David L. Dill, Alan J. Hu, and Howard Wong-Toi. Checking for language inclusion using simulation preorders. In Kim G. Larsen and Arne Skou, editors, *CAV 1991*, vol. 575 of *LNCS*, pp. 255–265, 1991.

[EH00] Kousha Etessami and Gerard Holzmann. Optimizing Büchi automata. In Catuscia Palamidessi, editor, *CONCUR 2000*, vol. 1877 of *LNCS*, pp. 153–167, 2000.

[EJ88] E. Allen Emerson and Charanjit S. Jutla. The complexity of tree automata and logics of programs (extended abstract). In *FoCS 1988*, pp. 328–337. 1988. IEEE.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. Tree automata, mu-calculus and determinacy. In *FoCS 1991*, pp. 368–377, San Juan, Puerto Rico, Oct. 1991. IEEE.

[ESW01] Kousha Etessami, Rebecca Schuller, and Thomas Wilke. Fair simulation relations, parity games, and state space reduction for Büchi automata. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *ICALP 2001*, vol. 2076 of *LNCS*, pp. 694–707, 2001.

[Ete02] Kousha Etessami. A Hierarchy of Polynomial-Time Computable Simulations for Automata. CONCUR 2002, to appear.

[FW02] Carsten Fritz and Thomas Wilke. Simulation relations for alternating Büchi automata. Extended version of tech. rep. 2019, Institut für Informatik, CAU Kiel, July 2002. URL: `http://www.informatik.uni-kiel.de/~fritz/TechRep2019_ext.ps`.

[GH82] Yuri Gurevich and Leo Harrington. Trees, automata, and games. In *14th ACM Symp. on the Theory of Computing*, pp. 60–65, 1982.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Co., San Francisco, 1979.

[GO01] Paul Gastin and Denis Oddoux. Fast LTL to Büchi automata translation. In G. Berry, H. Comon, and A. Finkel, editors, *CAV 2001*, vol. 2102 of *LNCS*, pp. 53–65, 2001.

[GPVW95] Rob Gerth, Doron Peled, Moshe Y. Vardi, and Pierre Wolper. Simple on-the-fly automatic verification of linear temporal logic. In *PSTV 1995*, pp. 3–18, Warsaw, Poland, June 1995. Chapman Hall.

[GBS02] Sankar Gurumurthy, Roderick Bloem, and Fabio Somenzi. Fair simulation minimization. CAV 2002, to appear.

[HHK95] Monika Henzinger Rauch, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *FoCS 1995*, pp. 453–462, 1995.

[HKR97] Thomas A. Henzinger, Orna Kupferman, and Sriram K. Rajamani. Fair simulation. In *CONCUR 1997*, vol. 1243 of *LNCS*, pp. 273–287, 1997.

[HR00] Thomas A. Henzinger and Sriram K. Rajamani. Fair bisimulation. In S. Graf and M. Schwartzbach, editors, *TACAS 2000*, vol. 1785 of *LNCS*, pp. 299–314, 2000.

[Jur00] Marcin Jurdziński. Small progress measures for solving parity games. In H. Reichel and S. Tison, editors, *STACS 2000*, vol. 1770 of *LNCS*, pp. 290–301, 2000.

[KVW00] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

[LN01] Martin Leucker and Thomas Noll. Truth/SLC - A parallel verification platform for concurrent systems. In G. Berry, H. Comon, and A. Finkel, editors, *CAV 2001*, vol. 2102 of *LNCS*, pp. 255–259, 2001.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer, Boston, 1993.

[MH84] S. Miyano and T. Hayashi. Alternating finite automata on ω-words. *Theoretical Computer Science*, 32:321–330, 1984.

[Mil71] Robin Milner. An algebraic definition of simulation between programs. In D. C. Cooper, editor, *Proc. of the 2nd Int. Joint Conf. on Artificial Intelligence*, pp. 481–489, London, UK, September 1971. William Kaufmann.

[MSS88] David E. Muller, Ahmed Saoudi, and Paul E. Schupp. Weak alternating automata give a simple explanation of why most temporal and dynamic logics are decidable in exponential time. In *LICS 1988*, pp. 422–427, IEEE Computer Society, 1988.

[SB00] Fabio Somenzi and Roderick Bloem. Efficient Büchi automata from LTL formulae. In E. Allen Emerson and A. Prasad Sistla, editors, *CAV 2000*, vol. 1855 of *LNCS*, pp. 248–263, 2000.

[Var94] Moshe Y. Vardi. Nontraditional applications of automata theory. In *Theoretical Aspects of Computer Software*, vol. 789 of *LNCS*, pp. 575–597, 1994.

[VW86] Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification. In Dexter Kozen, editor, *LICS 1986*, pp. 332–344, Cambridge, Mass., 16–18 June 1986.

[VW94] Moshe Y. Vardi and Pierre Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

Algorithmic Combinatorics Based on Slicing Posets

Vijay K. Garg[1,2,⋆]

[1] Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712-1084, USA
garg@ece.utexas.edu
[2] Department of Computer Science & Engineering
Indian Institute of Technology,
Kanpur, India

Abstract. We show that some recent results in slicing of a distributed computation can be applied to developing algorithms to solve problems in combinatorics. A combinatorial problem usually requires enumerating, counting or ascertaining existence of structures that satisfy a given property B. We cast the combinatorial problem as a distributed computation such that there is a bijection between combinatorial structures satisfying B and the global states that satisfy a property equivalent to B. We then apply results in slicing a computation with respect to a predicate to obtain a small representation of only those global states that satisfy B. The slicing results are based on a generalization of Birkhoff's Theorem of representation of finite distributive lattices. This gives us an efficient (polynomial time) algorithm to enumerate, count or detect structures that satisfy B when the total set of structures is large but the set of structures satisfying B is small. We illustrate our techniques by analyzing problems in integer partitions, set families, and set of permutations.

1 Introduction

Consider the following combinatorial problems:
(Q1) Count the number of subsets of the set $[n]$ (the set $\{1 \ldots n\}$) which have size m and do not contain any consecutive numbers.
(Q2) Enumerate all integer partitions less than $(\lambda_1, \lambda_2, \ldots, \lambda_n)$ in which the first part is equal to the second part.
(Q3) Give the number of permutations of $[n]$ in which $i \leq j$ implies that the number of inversions of i is atmost the number of inversions of j.

Our goal in this paper is to show how such problems can be solved *mechanically* and *efficiently* for any fixed values of the parameters n and m. It is important to note that someone trained in combinatorics may be able to solve all of these problems efficiently. Our emphasis is on techniques that can be applied

⋆ Supported in part by the NSF Grants ECS-9907213, CCR-9988225, Texas Education Board Grant ARP-320, an Engineering Foundation Fellowship, and an IBM grant.

mechanically. On the other hand, for the fixed values of n and m, all the sets above are finite and therefore all the problems can be solved mechanically. Our emphasis is on *efficiency*. To be more precise, let L be a large set of combinatorial structures (for example, all subsets of $\{1 \ldots n\}$ of size m, all permutations of $[n]$ etc.) Each combinatorial problem requires enumerating, counting, or searching the subset of structures that satisfy a given property B. Call this set $L_B \subseteq L$. For example, in the problem (Q1), L is the set of all subsets of $[n]$ of size m and L_B is the set of all subsets that do not contain any consecutive numbers from $[n]$. For any fixed set of parameters m and n, the size of L is large but finite, enabling one to enumerate all possible structures and then to check each one of them for the property B. This approach results in an algorithm that requires time proportional to the set L which is exponential in n (or m). This paper proposes a technique that provides answers to some combinatorial problems in polynomial time and for others, such as those involving enumeration, in time proportional to the size of L_B (and not L). Our technique is applicable whenever B satisfies a property called *regularity* and we give several examples of regular B in this paper.

To explain our technique, we use the term *small* to mean polynomial in n and m, and *large* to mean exponential in n or m. Thus, the set L is large. We first build a *small* structure P such that all elements of P can be generated by L. Second, we compute a *slice* of P with respect to B, denoted by P_B, such that P_B generates all elements of L_B and when B is regular only those elements. P_B is a small structure and can be efficiently analyzed to answer questions about L_B. For regular B, one could simply enumerate all elements of L_B from P_B.

Our approach is based on some recent results on *slicing* a distributed computation with respect to a predicate B [GM01,MG01]. Informally, a slice of a computation with respect to a predicate B is a subcomputation with the least number of global states that contains all global states that satisfy B. Slicing, in turn, is based on Birkhoff's Theorem of representation of finite distributive lattices [DP90]. The small structure P is a directed graph representing a distributed computation on n processes. The set of all (consistent) global states of the computation is the large structure L.

Consider any predicate B defined on L, or equivalently, the a subset L_B of L. B is called regular if L_B is a sublattice of L. From Birkhoff's theorem we know that there exists a poset that generates L_B. We show that every sublattice of L can be generated by a poset that can be obtained by adding edges to the poset P. Note that when edges are added to the graph of a poset cycles may form. In this case we simply consider the poset of strongly connected components in the graph. We denote the small structure obtained after adding edges to P as P_B. Now P_B can be used to enumerate elements in L_B, or to analyze the number of elements in L_B. Many algorithms have been proposed to enumerate ideals of a poset; for example by Steiner[Ste86] and Squire[Squ95]. In distributed computing, the algorithms to explore the global state lattice address the identical problem (see [CM91,AV01,Gar02]). Determining the count of the elements in L_B

given P_B is #P-complete for general posets [PB83] but can be done efficiently for 2-dimensional posets[Ste84].

We apply these ideas to many traditional problems in combinatorics. Due to the lack of space, all proofs in this paper are omitted and the interested reader can consult the technical report available at the author's website.

2 Notation and Definitions

A pair (X, P) is called a partially ordered set or poset if X is a set and P is a reflexive, antisymmetric, and transitive binary relation on X. We simply write P as a poset when X is clear from the context. We say $e \leq f$ in P when $(e, f) \in P$ and that f covers e if $e < f$ and there is no g such that $e < g < f$. Let $e, f \in X$ with $e \neq f$. If either $e < f$ or $f < e$, e and f are *comparable*. On the other hand, if neither $e < f$ nor $f < e$, then e and f are *incomparable*. A poset (X, P) is called *chain* if every distinct pair of points from X is comparable in P. Similarly, a poset is an *antichain* if every distinct pair of points from X is incomparable in P.

(X, P) and (Y, Q) are isomorphic, if there exists a $1-1$ and onto map $F : X \longrightarrow Y$ such that $c \leq d$ in P if and only if $F(c) \leq F(d)$ in Q. A poset (X, Q) is an extension of (X, P) if for all $e, f \in X$, $e < f$ in P implies $e < f$ in Q. (X, Q) is a *linear extension* if it is an extension of (X, P) and is a chain.

A *lattice* is a poset L such that for all $x, y \in L$, the least upper bound of x and y exists, called the *join* of x and y (denoted by $x \sqcup y$); and the greatest lower bound of x and y exists, called the *meet* of x and y (denoted by $x \sqcap y$). A *sublattice* is a subset of L closed under join and meet. A sublattice for which there exists two elements c and d such that it includes all x which lie between c and d (i.e. $c \leq x \leq d$) is called an *interval lattice* and denoted by $[c, d]$. A lattice L is *distributive* if for all $x, y, z \in X$: $x \sqcap (y \sqcup z) = (x \sqcap y) \sqcup (x \sqcap z)$.

Let (X, P) be a poset and let $G \subseteq X$. G is called an *order ideal* in (X, P) if $e \in G$ whenever $f \in G$ and $e \leq f$ in P. Consider the poset in Fig. 1(a). The set $\{b, d\}$ is an order ideal. The set $\{a, c\}$ is not because it includes c but does not include b which is smaller than c. We simply use *ideal* for order ideal in this paper. Let L denote the family of all ideals of P. Define a partial order on L by $G \leq H$ in L if and only if $G \subseteq H$. It is well known that the set of ideals forms a distributive lattice and conversely every finite distributive lattice can be constructed in this manner. Fig. 1(a) shows a poset and Fig. 1(b) its lattice of ideals. Given a finite distributive lattice L, one can determine the poset that generates L as follows. An element $e \in L$ is *join-irreducible* if it cannot be written as joins of two elements different from e (i.e., there is exactly one edge coming into the element, see Fig. 1(b)). For any $e \in P$, let $J(e)$ denote the least ideal in L that contains e. It is easy to show that $J(e)$ is join-irreducible. Let $J(L)$ denote the set of all join-irreducible elements in L. Birkhoff's theorem states that any finite distributive lattice L is isomorphic to the set of ideals of the poset $J(L)$ (and dually, any finite poset P is isomorphic to join-irreducible elements of the set of ideals of P). Meet-irreducible elements of L can be defined in an analogous

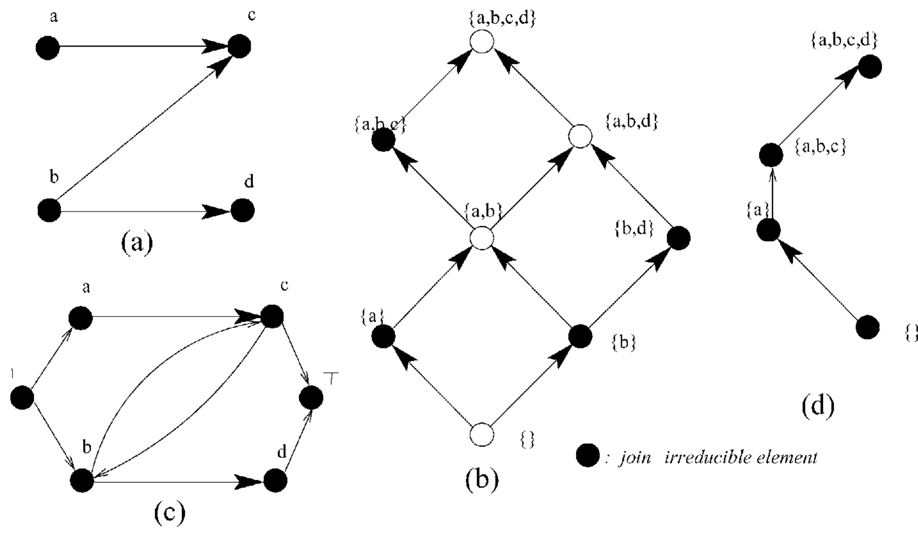


Fig. 1. (a) a partial order (b) the lattice of ideals (c) A slice (d) ideals of the slice

manner. $M(f)$, the greatest ideal that does not contain f, is meet-irreducible. The set of all meet-irreducible elements of L are denoted by $M(L)$ and Birkhoff's theorem can also be stated using $M(L)$.

In this paper, P and posets derived from P will serve as the small structures, and L and sublattices of L will serve as the large structures. We are usually interested in $L_B \subseteq L$, containing ideals of L that satisfy a given predicate B. Instead of enumerating L and checking for predicate B, we use P and B to derive a structure P_B that generates L_B.

For counting the number of elements in L and its sublattices, we use $N(P)$ to denote the number of ideals of the poset P. Since our interest is in efficient calculation of $N(P)$, we will restrict the *dimension* of the partial order generating the lattice. The dimension of a poset (X, P), denoted by $dim(X, P)$, is the least positive integer t for which there exists a family $\{C_1, C_2, \ldots, C_t\}$ of linear extensions of P (total orders compatible with P) such that $P = \cap_{i=1}^{t} C_i$. Determining whether a poset P with n points is 2-dimensional and isomorphism testing for 2-dimensional orders can be done in $O(n^2)$ time [Spi85]. All the posets used in this paper are 2-dimensional. The reader is referred to [Tro92] for dimension theory of posets. The following lemma shows that the number of ideals of a poset can be calculated efficiently for series-parallel posets (a special case of 2-dimensional posets) [FLST86]. For generalization to counting ideals of two dimensional posets see [Ste84].

Lemma 1 (Counting Lemma). *(1) If Q is an extension of P then $N(Q) \leq N(P)$. (2) Let $P + Q$ be the disjoint union (or direct sum) of posets P and Q (see [DP90]). Then, $N(P + Q) = N(P)N(Q)$. (3) Let $P \oplus Q$ be the ordinal sum of posets P and Q[DP90]. Then, $N(P \oplus Q) = N(P) + N(Q) - 1$. (4) Assume* *that P can be decomposed into the least number of chains $C_1, C_2, \ldots C_n$. Then $N(P) \leq \prod_{i=1}^{n}(|C_i| + 1)$. When each chain is at most m in length, we get that $N(P) \leq (m+1)^n$.*

For some examples, instead of enumerating all ideals of a poset we may be interested in enumerating or counting ideals in a certain *level set*. To define *level sets*, first define a poset to be *ranked* if for each element $x \in P$, one can assign a non-negative integer, $rank(x)$, such that if y covers x, then $rank(y) = rank(x) + 1$. The set of all elements in P with rank i are called it level set with rank i. Every distributive lattice is a ranked poset [Sta86].

3 Our Model

Traditionally the duality is expressed between finite posets and finite distributive lattices. In this paper, we are interested in producing structures that generate subsets of the finite distributive lattice. It is more convenient to use directed graphs instead of posets for this purpose because, as shown later, we can get sublattices by simply adding edges to the original directed graph.

The notion of ideals can be extended to graphs in a straightforward manner. A subset of vertices of a directed graph is an *ideal* iff the subset contains a vertex only if it contains all its incoming neighbors. Observe that an ideal either contains all vertices in a strongly connected component or none of them. Let $\mathcal{I}(P)$ denote the set of ideals of a directed graph P. Observe that the empty set $\emptyset$ and the set of all vertices trivially belong to $\mathcal{I}(P)$. We call them *trivial* ideals. It is shown in [MG01] that given a directed graph P, $\langle \mathcal{I}(P); \subseteq \rangle$ forms a distributive lattice.

Now assume that we are interested in the set of ideals that satisfy a predicate B. We will be interested in deriving a graph such that its ideals capture this set. A small difficulty is that every graph has at least two trivial ideals and therefore we cannot capture the case when the set of ideals satisfying B is empty. To address this problem, we add to the graph two additional vertices $\bot$ and $\top$ such that $\bot$ is the smallest vertex and $\top$ is the largest vertex. This ensures that any nontrivial ideal will contain $\bot$ and will not contain $\top$. As a result, every ideal of a computation in the traditional model is a nontrivial ideal of the computation in our model and vice versa. We will deal with only nontrivial ideals from now on.

Fig. 1(c) shows the directed graph and its nontrivial ideals. The directed graph in Fig. 1(c) is derived from Fig. 1(a) by adding an edge from c to b and adding two additional vertices $\bot$ and $\top$. The resulting set of nontrivial ideals is a sublattice of Fig. 1(d). In the figure, we have not shown $\bot$ in the ideals because it is implicitly included in every nontrivial ideal. This example illustrates the main steps in our technique. Fig. 1(a) is a small structure that generates the large structure Fig. 1(b). We are interested in enumerating or counting the set of ideals that satisfy the property “the ideal contains b whenever it contains c.” To generate such ideals it is sufficient to add an edge from c to b. Fig. 1(c) shows the small structure that generates all the ideals of interest to us. Fig. 1(c) will be called the slice of Fig. 1(a) with respect to the predicate B. The formal definition of a slice is given in the next section.

4 Slices and Regular Predicates

We denote the slice of a directed graph P with respect to a predicate B by $slice(P, B)$. The $slice(P, B)$ is a graph derived from P such that all the ideals in $\mathcal{I}(P)$ that satisfy B are included in $\mathcal{I}(slice(P, B))$. Note that the slice may include some additional ideals which do not satisfy the predicate. Formally,

Definition 1 (Slice [MG01]). *A slice of a graph P with respect to a predicate B is the directed graph obtained from P by adding edges such that (1) it contains all the ideals of P that satisfy B and (2) of all the graphs that satisfy (1), it has the least number of ideals.*

It is shown in [MG01] that the slice exists and is unique for every predicate. Computing slices for predicates in general is NP-hard but one can efficiently compute slices for *regular* predicates.

Definition 2 (Regular Predicates [GM01]). *A predicate is regular if the set of ideals that satisfy the predicate forms a sublattice of the lattice of ideals.*

Equivalently, a predicate B is *regular* with respect to P if it is closed under $\sqcup$ and $\sqcap$.

We now show that regular predicates can be decomposed into simpler structures called *simple* predicates. Our motivation is that computing slices for simple predicates is easy.

Definition 3 (Simple Predicates). *A predicate B is* simple *if there exists $e, f \in P$ such that $\forall G \in \mathcal{I}(P): \quad B(G) \equiv ((f \in G) \Rightarrow (e \in G))$*

Denote this predicate by $S(e, f)$. Thus, a simple predicate is of the form: G satisfies B iff whenever it includes f it includes e. We first show a useful property of simple predicates.

Lemma 2. *A simple predicate $S(e, f)$ partitions the lattice of ideals into two sublattices. Moreover, $\neg S(e, f)$ is equivalent to the interval lattice $[J(f), M(e)]$.*

In Fig. 1(c), our predicate is $S(c, b)$. The sublattice for $S(c, b)$ is shown in Fig. 1(d). Its complement, the set of ideals $[\ \{b\}, \{a, b\}, \{b, d\}, \{a, b, d\}\ ]$ is also a sublattice. We now show an easy test that indicates whether a regular predicate B is stronger than $S(e, f)$. Let $J_B(e)$ denote the least ideal that includes e and satisfies B. Since the predicate B is regular and the predicate "the ideal includes e" is also regular, it follows that $J_B(e)$ is well defined.

Lemma 3. *For regular B and any e,f*

$e \in J_B(f) \quad \equiv \quad J_B(e) \subseteq J_B(f) \quad \equiv \quad B \Rightarrow S(e, f)$

We now turn our attention to characterizing the set of ideals that satisfy B.

Lemma 4. *An ideal G satisfies a regular predicate B iff $\forall f \in G : J_B(f) \subseteq G$.*

We now provide a *decomposition* theorem for regular predicates.

Theorem 1. *For any regular predicate B, let $E_B = \{(e,f) | B \Rightarrow S(e,f)\}$. Then, $B = \bigwedge_{(e,f)\in E_B} S(e,f)$.*

From the decomposition theorem and properties of simple predicates we get that B is a regular predicate *iff* it can be expressed as a conjunction of simple predicates. As a corollary (by applying De Morgan's and using the result about complement of simple predicates), we also get the following Rival's theorem [Riv73].

Corollary 1. *A complement of a sublattice can be expressed as a union of interval lattices of the form $[c,d]$ where c is a join-irreducible element and d is a meet-irreducible element.*

This also implies that there are $O(2^{n^2})$ distributive lattices on n points. Every distributive lattice is a sublattice of the boolean lattice on n elements and therefore equivalent to a regular predicate. By the decomposition theorem, a regular predicate is a conjunction of at most $O(n^2)$ simple predicates. Note that there can be as many as $O(2^{2^n})$ subsets of the boolean lattice but only very few of them are sublattices.

Now obtaining slices for a regular predicate B is an easy task. We simply add edge (e,f) to the graph of P for every simple predicate $S(e,f)$ such that $B \Rightarrow S(e,f)$. Therefore, we have

Theorem 2. *Let P be a directed graph. Let Q be a directed graph obtained by adding edges to P. Then, $\mathcal{I}(Q)$ is a sublattice of $\mathcal{I}(P)$. Conversely, every sublattice of $\mathcal{I}(P)$ is generated by some directed graph Q obtained from P by adding edges.*

Suppose that the poset P has n chains each of size at most m. Observation that if $f \leq g$ in poset P, then for any e, $S(e,f)$ implies $S(e,g)$. Therefore, for any event e there are at most n simple predicates (at most one for every chain) as part of a regular predicate. We conclude that every regular predicate can be expressed as conjunction of at most n^2m simple predicates.

5 Application to Combinatorics

In this section we give several examples of slicing posets.

1. Boolean Algebra and Set Families

Let X be a ground set on n elements. By using $\subseteq$ as the order relation on the power set of X, we can view it as a distributive lattice L. This lattice (called boolean lattice) has $n+1$ levels and each level set of rank k in the boolean lattice corresponds to $\binom{n}{k}$ sets of size k. L is generated by the directed graph in Fig. 2(a) which can also be interpreted as a distributed computation n processes $\{P_1, \ldots P_n\}$. Each process P_i executes a single event e_i. It is easy to verify that there is a bijection between every nontrivial global state of the computation and a subset of X.

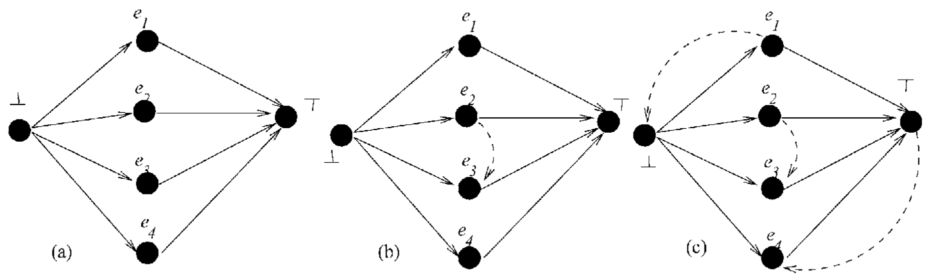


Fig. 2. Graphs and slices for generating subsets of X

Now consider all subsets of X such that if they include e_i then they also include e_j. To obtain the slice with respect to this predicate we just need to add an edge from e_j to e_i. Fig. 2(b) shows the slice with respect to the predicate that if e_3 is included then so is e_2. To ensure the condition that e_i is always included, we simply add an edge from e_i to $\perp$ and to ensure that e_i is never included in any subset, we add an edge from $\top$ to e_i. Fig. 2(c) shows the slice which gives all subsets that always contain e_1, never contain e_4 and contain e_2 whenever they contain e_3.

As an application, we now solve some combinatorial problems. Let n be even. We are required to calculate the total number of subsets of $[n]$ which satisfy the property that if they contain any odd integer i, then they also contain $i + 1$ (or equivalently, compute the number of ways to select groups from $n/2$ couples such that a wife is always accompanied by her husband in the group although a husband may not be accompanied by his wife). Although this problem can be solved directly by a combinatorial argument, we show how our method can be applied. First construct the poset which generates all the subsets of $[n]$. It is Fig. 2(a) in this case. Now define the subset of interest by a predicate B. For any subset G of $[n]$, Let $B(G)$ to be true if G contain $i+1$ whenever it contains any odd integer i. From our discussion of regular predicates, it is clear that B is regular and equivalent to $S(e_2, e_1) \wedge S(e_4, e_3) \ldots \wedge S(e_n, e_{n-1})$. To compute the slice, it is sufficient to add an edge from e_{i+1} to e_i for odd i. The slice consists of $n/2$ chains each with exactly 2 events (ignoring $\perp$ and $\top$). From the counting lemma (Lemma 1), it follows that the total number of ideals is $(2+1)^{n/2} = 3^{n/2}$. The reader should note that for any fixed value of n, the problem can be solved by a computer automatically and efficiently (because the slice results in a series-parallel poset).

2. Set Families of Size k

It is important to note that regularity of B is dependent upon the lattice structure of L. For example, in many applications of set families, we are interested in sets of a fixed size k. The predicate B that the ideal is of size k is not regular. However, by considering alternative posets, this set family can still be analyzed. Fig. 3 shows a computation such that all the subsets of X of size k are its ideals. For clarity, we have not drawn $\top$ and $\perp$ in the figure.

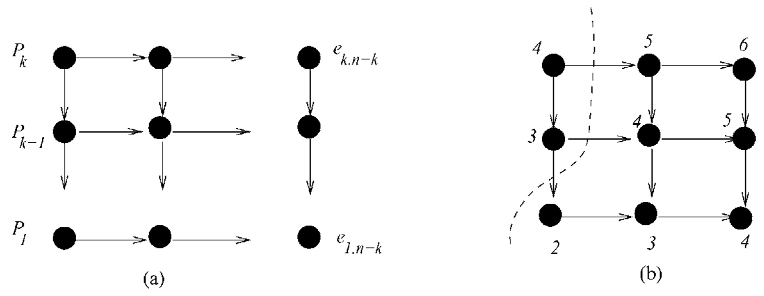


Fig. 3. (a) Graphs for subsets of X of size k (b) Example when $n = 6$ and $k = 3$

There are k processes in this computation and each process executes $n - k$ events. By the structure of the computation, if in a global state P_i has executed j events, then P_{i+1} must have also executed at least j events. The correspondence between subsets of X and global states can be understood as follows. If process P_i has executed t events in the global state, then the element $t + i$ is in the set Y. Thus process P_1 chooses a number from $1 \ldots n - k + 1$ (because there are $n - k$ events); process P_2 chooses the next larger number and so on. It can also be easily verified that the poset in Fig. 3(a) is a 2-dimensional poset and that there are $\binom{n}{k}$ ideals of this poset. Fig. 3 gives an example of the computation for subsets of size 3 of the set [6]. The global state, or the ideal, shown corresponds to the subset $\{1, 3, 4\}$.

Now let us apply our theory to the first combinatorial problem (Q1) mentioned in the introduction. Assume that we are interested in counting all subsets of n of size k which do not have any consecutive numbers. In this example, G satisfies B if whenever P_i has t events in G, P_{i+1} has at least $t + 1$ events in G. This condition is regular and we can use Lemma 3 and Theorem 1 to compute the slice. (for every event f, we only need to determine whether $e \in J_B(f)$). Fig. 4 shows the slice which includes precisely such subsets. By collapsing all strongly connected components and by removing the transitively implied edges we get a graph which is isomorphic to the case when there are k processes and each process executes $n - k - (k - 1)$ events. Therefore, the total number of such sets is $\binom{n-k+1}{k}$. Again one can come up with a combinatorial argument to solve the problem (for example, see Theorem 13.1 and Example 13.1 in [vLW92]), but the slicing approach is completely mechanical.

The above construction can be generalized to multidimensional grids to obtain results on multinomials instead of binomials.

3. Integer Partitions and Young's Lattice

A k-tuple of positive integers $\lambda = (\lambda_1, \ldots, \lambda_k)$ is an integer partition of N if $\lambda_1 + \ldots + \lambda_k = N$ and for all i, $\lambda_i \geq \lambda_{i+1}$. The number of *parts* of λ is k. An example of partition of 10 into 3 parts is $(4, 3, 3)$. An integer partition can be visualized as a *Ferrers diagram* or an array of squares in decreasing order with λ_i squares in row i. The Ferrers diagram of the partition $(4, 3, 3)$ of 10 is shown

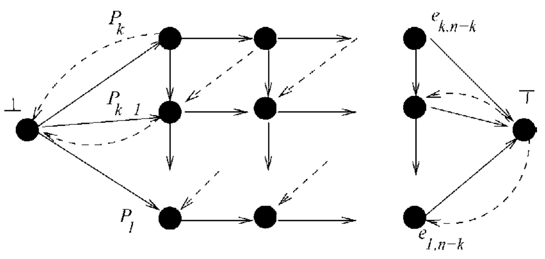


Fig. 4. Slice for the predicate "does not contain consecutive numbers"

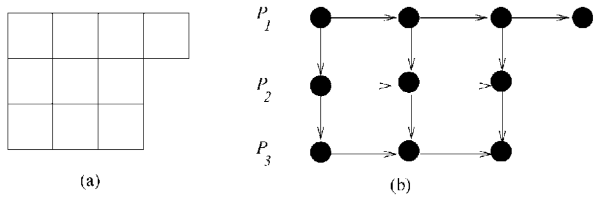


Fig. 5. (a) A Ferrer diagram (b) A graph for generating Young's lattice

in Fig. 5(a). A partition λ is contained in another partition δ if the number of parts of λ is at most that of δ and λ_i is less than or equal to δ_i for any i between 1 and the number of parts of λ. For example, $(3,3,1)$ is less than $(4,3,3)$. Fix any partition λ. The set of all partitions that are less than or equal to λ form the *Young's lattice* denoted by Y_λ.

We now apply our approach to Y_λ. Let the number of parts and the largest part in the partition λ be m and n respectively. Then we have a distributed computation of n processes with at most m events per process as shown in Fig. 5(b). P_i executes as many events as λ_i. It is clear that for any global state, the number of events executed by P_i is at least as many as executed by P_{i+1}. Clearly, the set of global states of the computation as in Fig. 5(b) is isomorphic to Young's lattice for the corresponding partition.

It follows that Young's lattice is distributive. One can see that the lattice of subsets of size k from the set of size n is a special case of Youngs's lattice when all λ_i's are equal. Therefore, the number of integer partitions whose Ferrers diagrams fit in a box of size k by $n-k$ is equal to $\binom{n}{k}$ (providing an alternate proof of Theorem 3.2 in [SW86]). Let $q(N,k,m)$ denote the number of partitions of N whose Ferrer's diagram fit in a box of size k by m. By summing up the sizes of all level sets, we get $\binom{n}{k} = \sum_{l=0}^{k(n-k)} q(l,k,n-k)$. Since the poset that generates corresponding Young's lattice is symmetric with respect to k and m, we get that $q(N,k,m)$ equals $q(N,m,k)$; and since the poset is dual of itself (i.e. we get back the same poset when all arcs are reversed) we also get that $q(N,k,m)$ equals $q(mk-N,k,m)$. All these results are well known and generally derived using Gaussian polynomials (see [vLW92]).

Now assume that we are interested in all those partitions such that their second component is some fixed value say b. It is easy to verify that partitions

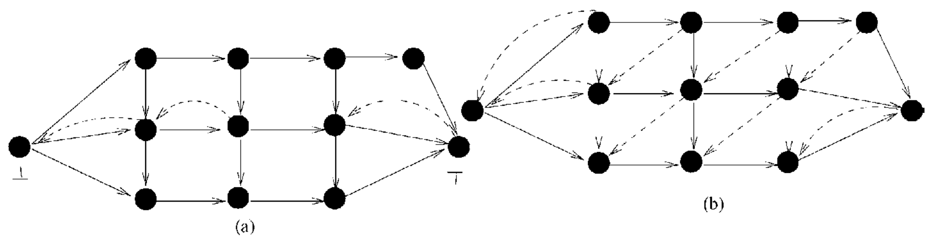


Fig. 6. (a)Slice for $\delta_2 = 2$ (b) Slice for "distinct parts"

$\delta \in Y_\lambda$ such that $\delta_c = b$ form a sublattice and therefore the condition $\delta_c = b$ is a regular predicate. Fig. 6(a) gives the slice of partitions in which $\delta_2 = 2$. Since the second part must be 2, we add edges to ensure that P_2 executes exactly 2 events. On collapsing the strongly connected components, transitively reducing the graph and applying counting lemma, we get that there are $(2+1)(2+1) = 9$ such partitions.

As another example assume that we are interested in all partitions less than λ which have distinct parts. Fig. 6(b) gives the slice. The graph is equivalent to that of subsets of size 3 from [5]. Hence, there are $\binom{5}{3}$ such partitions. Some other subsets of partitions discussed in the literature are "partitions with odd number of parts", "partitions with distinct odd parts," "partitions with even number of parts" etc. These are also regular predicates.

Now the reader may also see the solution for the second problem (Q2) mentioned in the introduction—enumerating all partitions in the Young's lattice Y_λ with first part equal to the second part. We simply define the predicate B on a partition δ to be true when δ_1 equals δ_2. It is clear that the predicate is closed under joins and meets and is therefore a regular predicate. One can draw the slice and conclude that the number of partitions δ in Y_λ satisfying $\delta_1 = \delta_2$ is equal to the number of partitions in Y_δ where $\delta = (\lambda_2, \lambda_3, \ldots, \lambda_k)$.

Note that the level set of rank N of Y_λ (where $\lambda = (\lambda_1, \lambda_2 \ldots, \lambda_t)$) corresponds to all partitions of N with at most t parts and the largest part at most λ_1. It follows that all partitions of N can be enumerated as the elements in level set of rank N of $Y_{(N,N,..N)}$.

4. Permutations

We first show a small computation that generates all permutations of n symbols. The computation consists of $n-1$ processes. Process P_i executes $i-1$ events. We use the notion of the inversion table[Knu98] to interpret the choices made by processes. The number of inversions of i in a permutation π is the number of symbols less than i that appear to the right of i in π. The way a permutation is generated from a global state is as follows. We begin the permutation by writing 1. P_1 decides where to insert the symbol 2. There are two choices. If we place 2 after 1, then we introduce zero inversions; otherwise we introduce one inversion. Proceeding in this manner we get that there is a bijection between the set of permutations and the global states.

It is easy to show that

Lemma 5. *All the following properties of permutations are regular. (1) The symbol $m < n$ has* at most *j inversions (for $j < m$). The total number of such permutations is $\frac{n!(j+1)}{m}$. (2) $i \leq j$ implies that i has at most as many inversions as j. The total number of such permutations is same as the number of integer partitions less than $(n-1, n-2, ..., 1)$.*

Further by computing the slice, we can also calculate the number of permutations satisfying B. The level set at rank k of the permutation lattice consists of all permutations with total number of inversions equal to k and therefore such permutations can be efficiently enumerated [Knu98,ER02].

References

AV01. S. Alagar and S. Venkatesan. Techniques to tackle state explosion in global predicate detection. *IEEE Transactions on Software Engineering*, 27(8):704 – 714, August 2001.

CM91. R. Cooper and K. Marzullo. Consistent detection of global predicates. In *Proc. of the Workshop on Parallel and Distributed Debugging*, pages 163–173, Santa Cruz, CA, May 1991. ACM/ONR.

DP90. B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order*. Cambridge University Press, Cambridge, UK, 1990.

ER02. S. Effler and F. Ruskey. A CAT algorithm for listing permutations with a given number of inversions. *Information Processing Letters*, 2002.

FLST86. U. Faigle, L. Lovász, R. Schrader, and Gy. Turán. Searching in trees, series-parallel and interval orders. *SIAM Journal on Computing*, 15(4):1075–1084, 1986.

Gar02. V. K. Garg. Detecting global predicates in distributed computations. Technical report, Parallel and Distributed Systems Laboratory, ECE Dept. University of Texas at Austin, September 2002. available at `www.ece.utexas.edu/~garg/pubs.html`.

GM01. V. K. Garg and N. Mittal. On slicing a distributed computation. In *21st International Conference on Distributed Computing Systems (ICDCS' 01)*, pages 322–329, Washington - Brussels - Tokyo, April 2001. IEEE.

Knu98. Donald E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming*. Addison-Wesley, Reading, MA, USA, second edition, 1998.

MG01. N. Mittal and V. K. Garg. Slicing a distributed computation: Techniques and theory. In *5th International Symposium on DIStributed Computing (DISC'01)*, pages 78 – 92, October 2001.

PB83. J.S. Provan and M. O. Ball. The complexity of counting cuts and of computing the probability that a graph is connected. *SIAM Journal on Computing*, 12:777–788, 1983.

Riv73. I. Rival. Maximal sublattices of finite distributive lattices. *Proc. Amer. Math. Soc.*, pages 417–420, 1973.

Spi85. Jeremy Spinrad. On comparability and permutation graphs. *SIAM Journal on Computing*, 14(3):658–670, 1985.

Squ95. M. Squire. *Gray Codes and Efficient Generation of Combinatorial Structures.* PhD Dissertation, Department of Computer Science, North Carolina State University, 1995.

Sta86. R. Stanley. *Enumerative Combinatorics Volume 1*. Wadsworth and Brookes/ Cole, Monterey, California, 1986.

Ste84. G. Steiner. Single machine scheduling with precedence constraints of dimension 2. *Math. Operations Research*, 9:248 – 259, 1984.

Ste86. G. Steiner. An algorithm to generate the ideals of a partial order. *Operations Research Letters*, 5(6):317 – 320, 1986.

SW86. D. Stanton and D. White. *Constructive Combinatorics*. Springer-Verlag, 1986.

Tro92. W.T. Trotter. *Combinatorics and Partially Ordered Sets: Dimension Theory*. The Johns Hopkins University Press, 1992.

vLW92. J.H. van Lint and R. M. Wilson. *A Course in Combinatorics*. Cambridge University Press, 1992.

Pattern Matching for Arc-Annotated Sequences

Jens Gramm[⋆], Jiong Guo[⋆⋆], and Rolf Niedermeier

Wilhelm-Schickard-Institut für Informatik, Universität Tübingen
Sand 13, D-72076 Tübingen, Fed. Rep. of Germany
{gramm,guo,niedermr}@informatik.uni-tuebingen.de

Abstract. A study of pattern matching for arc-annotated sequences is started. An $O(nm)$ time algorithm is given to determine whether a length m sequence with nested arc annotations is an arc-preserving subsequence of a length n sequence with nested arc annotations, called APS(NESTED,NESTED). Arc-annotated sequences and, in particular, those with nested arc structure are motivated by applications in RNA structure comparison. Our algorithm can be used to accelerate a recent fixed-parameter algorithm for LAPCS(NESTED,NESTED) and generalizes results for ordered tree inclusion problems. In particular, the presented dynamic programming methodology implies a quadratic time algorithm for an open problem posed by Vialette.

1 Introduction

Basic motivation. Pattern matching in strings is a core problem of computer science. It is of foundational importance in several application areas, the most recent one being computational biology. Numerous versions of pattern matching and related problems occur in practice, ranging in difficulty from linear time solvable to NP-hard problems. In this paper, we study a pattern matching problem that was originally motivated by RNA structure comparison and motif search, a topic that has recently received considerable attention [3,4,5,7,9]. Herein, we encounter a seemingly sharp border between (practically important) problem versions that we show to be efficiently solvable in quadratic time and slightly more general versions that turn out to be NP-complete.

Problem definition. We study pattern matching for *arc-annotated* sequences. Due to the problem motivation from computational biology, we use the terms "string" and "sequence" in a synonymous way. Note, however, that we clearly distinguish between the terms "substring" and "subsequence," the latter being the much more general term. For a sequence S, an *arc annotation* of S is a set of unordered pairs of positions in S. Lin *et al.* [7] argue that the biologically most important special case is the one of *nested* arc annotations. Here, one requires that no two arcs share an endpoint and no two arcs cross each other. We will also

[⋆] Supported by the Deutsche Forschungsgemeinschaft (DFG), research project OPAL (optimal solutions for hard problems in computational biology), NI 369/2-1.

[⋆⋆] Partially supported by the Deutsche Forschungsgemeinschaft (DFG), Zentrum für Bioinformatik Tübingen (ZBIT).

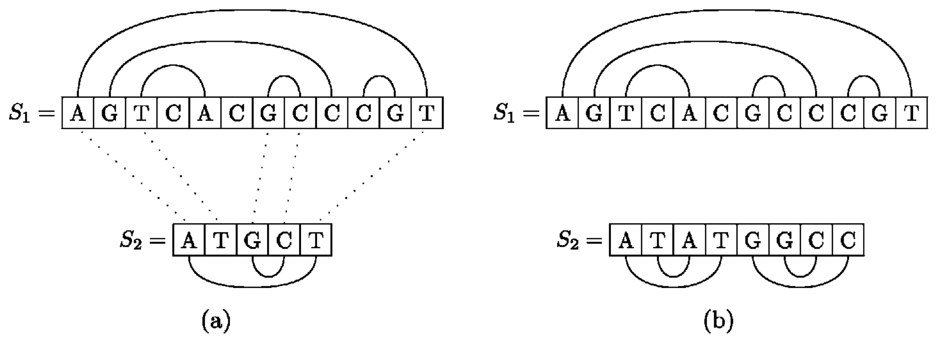

$S_1 =$ A G T C A C G C C C G T
$S_2 =$ A T G C T
(a)
$S_1 =$ A G T C A C G C C C G T
$S_2 =$ A T A T G G C C
(b)

Fig. 1. Input instances of APS(NESTED, NESTED). (a) Yes-instance, S_2 is an aps of S_1. (b) No-instance, S_2 is *not* an aps of S_1

Table 1. Survey of computational complexity for different versions of APS. Most NP-completeness results easily follow from results of Evans [4] for the corresponding LAPCS problems, or, at least, can be proven in a similar way in spirit to there. We omit the details here. The $O(nm)$ time algorithms are described in this paper. The complexity of APS(CROSSING,PLAIN) remains unclassified. We mention in passing that APS(CHAIN,PLAIN) can be solved in $O(n + m)$ time

APS(.,.)	UNLIMITED	CROSSING	NESTED	CHAIN	PLAIN
UNLIMITED	NP-complete [4]				
CROSSING	—	NP-complete [4]	NP-complete		?
NESTED	—	—	$O(nm)$		

consider the less restrictive *crossing* arc structure where the only requirement is that no two arcs share an endpoint. Furthermore, we exhibit the simpler arc structure *chain* which is a nested structure with nesting depth one. Finally, the term *plain* refers to sequences without arcs and the term *unlimited* refers to a completely unrestricted arc structure. Now, we are ready to define the pattern matching problems studied, namely the ARC-PRESERVING SUBSEQUENCE problem for various types of arc annotations: APS(TYPE1, TYPE2).
Input: An arc-annotated sequence S_1, $|S_1| = n$, with arc structure TYPE1 and an arc-annotated sequence S_2, $|S_2| = m$, with arc structure TYPE2.
Question: Does S_2 occur as an arc-preserving subsequence (aps) in S_1?
That is, if one deletes all but m letters from S_1 (thus, we assume $n \geq m$)—when deleting a letter at position i then *all* arcs with endpoint i are deleted—can one obtain S_2? Fig. 1(a) gives an example for a yes-instance and Fig. 1(b) gives an example for a no-instance of APS(NESTED,NESTED). Clearly, the problem specification only makes sense if arc structure TYPE1 comprises TYPE2.
Results. Our main result is that APS(NESTED,NESTED) can be solved in $O(nm)$ time. Table 1 surveys known and new results for various types of APS. In addition, we study a *modified* version of APS(UNLIMITED,NESTED) where the alphabet is unary, each base has to be endpoint of at least one arc, and we determine whether S_2 forms an *arc substructure (ast)* of S_1. Deriving an $O(nm)$ time al-

gorithm for this case, we answer an open question of Vialette [9]. Observe that in general APS(UNLIMITED,NESTED) is NP-complete (see Table 1). Due to the lack of space, several details had to be omitted.

Relations to previous work. There are basically two lines of research our results refer to. The first one is that of similar pattern matching problems and the other one is that of results (mostly NP-completeness, approximation, and fixed-parameter tractability) for the more general LONGEST ARC-PRESERVING COMMON SUBSEQUENCE problem (LAPCS). As to directly related pattern matching problems, the work perhaps most closely connected to ours is that of Vialette [9]. He studied very similar pattern matching problems also motivated by RNA secondary structure analysis. Although most of our results do not directly compare to his ones (because of the somewhat different models), our approach leads to an answer of one of his open questions asking for the algorithmic complexity (NP-complete vs. polynomially solvable) of the aforementioned modified case of APS(UNLIMITED,NESTED). Moreover, our work generalizes results achieved in the context of structured text databases for the so-called ordered tree inclusion problem [8]. Kilpeläinen and Mannila already presented quadratic time algorithms for a strict special case of APS(NESTED,NESTED). In their case, sequence information is not taken into account and we can easily derive their problem from ours. Bafna *et al.* [2], among other things, considered the corresponding arc-preserving *substring* problems.

As to LAPCS problems, we only briefly mention that most problems in that context become NP-complete [4,7] as, in particular, LAPCS(NESTED,NESTED). That is why researchers focussed on approximation (factor 2) [5] and fixed-parameter algorithms [1]. Notably, our algorithm for APS(NESTED,NESTED) can be used as a subprocedure to handle easy cases in the exact exponential time algorithm of [1]. Thus, a speed-up of this fixed-parameter algorithm can be achieved. Similar applications might be possible in the approximation context. Finally, note that for easier arc structures such as in LAPCS(NESTED,CHAIN) $O(nm^3)$ time algorithms have been developed [5]—for the special case APS(NESTED,CHAIN) beaten by our $O(nm)$ algorithm.

2 Preliminaries and Definitions

For a sequence S of length $|S| = n$, an *arc annotation* (or *arc set*) A of S is a set of pairs of numbers from $\{1, 2, \ldots, n\}$. Each pair $(i_l, i_r) \in A$ satisfies $i_l < i_r$ and connects the two *bases* $S[i_l]$ and $S[i_r]$ at positions i_l and i_r in S by an arc. In most cases, we will require that no two arcs share an endpoint, i.e., $(i_l, i_r), (i'_l, i'_r) \in A$ only if all i_l, i_r, i'_l, and i'_r are pairwisely distinct. Let S_1 and S_2 be two sequences with arc sets A_1 and A_2, respectively. If $S_1[i] = S_2[j]$ for $1 \leq i \leq |S_1|$ and $1 \leq j \leq |S_2|$ we refer to this as a *base match.* If S_2 is a subsequence of S_1 then it induces a one-to-one mapping M from $\{1, 2, \ldots, |S_2|\}$ to a subset of $\{1, 2, \ldots, |S_1|\}$, given by $M = \{\langle j, i_j \rangle \mid 1 \leq j \leq |S_2|, 1 \leq i_j \leq |S_1|\}$. We say that S_2 is an *arc-preserving subsequence (aps)* of S_1 if the arcs induced by M are preserved, i.e., for all $\langle j_l, i_l \rangle, \langle j_r, i_r \rangle \in M$: $(j_l, j_r) \in A_2 \Longleftrightarrow (i_l, i_r) \in A_1$.

In this paper, we study the ARC-PRESERVING SUBSEQUENCE problem (APS): Given an arc-annotated sequence S_1 and an arc-annotated pattern sequence S_2, the question is to determine whether S_2 is an aps of S_1. We use $|S_1| := n$ and $|S_2| := m$ if not mentioned otherwise. Depending on the arc structures of S_1 and S_2, several versions APS(TYPE1, TYPE2) can be defined where the arc structure of S_1 is TYPE1 and the arc structure of S_2 is TYPE2. An arc set has *nested* arc structure if no two arcs share an endpoint and no two arcs cross each other, i.e., for all (i_l^1, i_r^1), $(i_l^2, i_r^2) \in A$ it holds that $i_l^2 < i_l^1 < i_r^2$ iff $i_l^2 < i_r^1 < i_r^2$. In *plain* arc structures, the sequence has no arcs at all, *chain* arc structures have nested structure with nesting depth one, *crossing* refers to arc structures where the only requirement is that no two arcs share an endpoint, and *unlimited* refers to completely arbitrary arc structures.
Under the restriction that the alphabet is unary, we define a new problem which is related to aps. We assume arc-annotated sequences (S_1, A_1) and (S_2, A_2) such that every base in S_1 and every base in S_1 is endpoint of an arc. A sequence $i_1, \ldots, i_{|S_2|} \in \{1, \ldots, |S_1|\}$ with $i_1 < \cdots < i_{|S_2|}$ defines a mapping $M = \{\langle j, i_j\rangle \mid 1 \le j \le |S_2|, 1 \le i_j \le |S_1|\}$. Then, we say that S_2 is an *arc substructure (ast)* of S_1 if there is a mapping M such that arcs in A_2 are matched to arcs in A_1, i.e., for all $\langle j_l, i_l\rangle, \langle j_r, i_r\rangle \in M$: $(j_l, j_r) \in A_2 \Longrightarrow (i_l, i_r) \in A_1$. Then, the ARC-SUBSTRUCTURE problem (AST) is to determine whether S_2 is an ast of S_1. Analogously to APS, we can define AST(TYPE1, TYPE2), where the arc structures of S_1 and S_2 are, e.g., unlimited or nested.
For each arc $(i_l, i_r) \in A_1$, we define a set $I_1^{(i_l,i_r)}$ (analogously $I_2^{(j_l,j_r)}$ for $(j_l, j_r) \in A_2$) which contains positions of the bases that are inside arc (i_l, i_r) but not inside any arcs that are inside (i_l, i_r),

$$I_1^{(i_l,i_r)} = \{\ i \mid i_l < i < i_r\}\ \setminus \bigcup_{\substack{(i'_l,i'_r)\in A_1 \\ \wedge\ i_l<i'_l<i'_r<i_r}} \{\ i' \mid i'_l < i' < i'_r\}.$$

If A_1 has a *nested* arc structure then the sets $I_1^{(i_l,i_r)}$ for different arcs are disjoint. We define I_1 as the set of positions of endpoints of the outermost arcs in A_1 and of positions of all bases which are not inside any arcs in A_1. An arc $(i_l, i_r) \in A_1$ is a *matching arc* for an arc $(j_l, j_r) \in A_2$ if the corresponding endpoints of the two arcs are the same, i.e., $S_1[i_l] = S_2[j_l]$ and $S_1[i_r] = S_2[j_r]$, and $S_2[j_l + 1, j_r - 1]$ is an arc-preserving subsequence of $S_1[i_l + 1, i_r - 1]$. An *innermost matching arc* $(i_l, i_r) \in A_1$ for $(j_l, j_r) \in A_2$ is an arc which is a matching arc for (j_l, j_r) such that there is no arc inside (i_l, i_r) that is also a matching arc for (j_l, j_r).

3 APS(NESTED,PLAIN)

An instance of APS(NESTED, PLAIN) is given by (S_1, A_1) and (S_2, A_2), where S_1 has a nested arc structure A_1 while there is no arc in S_2, i.e., $A_2 = \emptyset$. We assume that $n \ge m$ since, otherwise, S_2 cannot be a subsequence of S_1. To solve the problem, we construct a dynamic programming table T of size $O(|A_1|m)$. Each

Computation of maxaps$(S_1[i_1, i_2], S_2[j_1, j_2])$:

- If $i_1 > i_2$ or $j_1 > j_2$ then maxaps$(S_1[i_1, i_2], S_2[j_1, j_2]) := j_1 - 1$.
- If $i_1 = i_2$ then maxaps$(S_1[i_1, i_1], S_2[j_1, j_2]) :=$
  - j_1, if $S_1[i_1] = S_2[j_1]$;
  - $j_1 - 1$, otherwise.
- If $j_1 = j_2$ and $i_1 < i_2$ then maxaps$(S_1[i_1, i_2], S_2[j_1, j_1]) :=$
  - j_1, if $S_1[i_1] = S_2[j_1]$;
  - maxaps$(S_1[i_1 + 1, i_2], S_2[j_1, j_1])$, otherwise.
- If $i_1 < i_2$ and $j_1 < j_2$ then maxaps$(S_1[i_1, i_2], S_2[j_1, j_2]) :=$
  - maxaps$(S_1[i_1 + 1, i_2], S_2[j_1 + 1, j_2])$, if $S_1[i_1] = S_2[j_1]$ and $S_1[i_1]$ is not an endpoint of an arc;
  - maxaps$(S_1[i_1 + 1, i_2], S_2[j_1, j_2])$, if $S_1[i_1] \neq S_2[j_1]$ and $S_1[i_1]$ is not an endpoint of an arc;
  - maxaps$(S_1[i_r + 1, i_2], S_2[T[i_l, j_1] + 1, j_2])$, if $S_1[i_1]$ is the left endpoint of an arc $(i_l, i_r) \in A_1$, i.e., $i_1 = i_l$.

Fig. 2. Recursive definition of maxaps

arc in A_1 corresponds to a row of this table and each position of S_2 corresponds to a column. We refer to the table entries corresponding to an arc $(i_l, i_r) \in A_1$ by $T(i_l, j)$, where j is an arbitrary position in S_2. Entry $T(i_l, j)$ is defined to contain the rightmost position $j' \geq j$ in S_2 such that $S_2[j, j']$ is an arc-preserving subsequence of $S_1[i_l, i_r]$ (or $j - 1$ if no such j' exists). In order to compute table T, we process the arcs of A_1 in the order of their right endpoints. The nested arc structure of A_1 implies that, when processing an arc $(i_l, i_r) \in A_1$, all arcs $(i'_l, i'_r) \in A_1$ inside (i_l, i_r), i.e., $i_l < i'_l < i'_r < i_r$, have already been processed. Thus, we process the arcs from inside to outside and from left to right.
We divide the algorithm into two main phases. The first phase computes the table entries corresponding to arcs in A_1 ordered by the arcs' right endpoints. When processing an arc $(i_l, i_r) \in A_1$, we use the table entries corresponding to the arcs directly inside (i_l, i_r). The second phase deals with those parts of S_1 which are outside all arcs. Here, we use the table entries corresponding to the outermost arcs in A_1. Thereby, we determine whether S_2 is an arc-preserving subsequence of S_1. In both phases, we make use of function maxaps, where maxaps$(S_1[i_1, i_2], S_2[j_1, j_2])$ returns the largest j', $j_1 \leq j' \leq j_2$, such that $S_2[j_1, j']$ is an arc-preserving subsequence of $S_1[i_1, i_2]$ (or $j_1 - 1$ if no such j' exists). Fig. 2 shows how to compute maxaps. Note that maxaps is well-defined since when we compute maxaps$(S_1[i_1, i_2], S_2[j_1, j_2])$ then all entries in T corresponding to arcs in A_1 with both endpoints inside $S_1[i_1, i_2]$ have already been computed before. An outline of the whole algorithm in pseudo-code is given in Fig. 3. An entry of table T, corresponding to $(i_l, i_r) \in A_1$, is computed by

$$T(i_l, j) := \max\{\text{maxaps}(S_1[i_l, i_r - 1], S_2[j, m]), \text{maxaps}(S_1[i_l + 1, i_r], S_2[j, m])\}.$$

By this way of computing $T(i_l, j)$, we match a longest possible substring starting at $S_2[j]$ to $S_1[i_l, i_r - 1]$ or $S_1[i_l + 1, i_r]$ and, by this means, we make sure that the

```
Procedure aps_np
Input: Sequence $S_1$ with nested arc structure
       and pattern sequence $S_2$ with no arcs;
Global variable: Array of int $T[n][m]$;
    /* (1) Processing the arcs in $A_1$ */
    for each $(i_l, i_r) \in A_1$ (ordered by their right endpoints) do
       for $j = 1$ to $m$ do
          $T(i_l, j) := \max \left\{ \begin{array}{l} \texttt{maxaps}(S_1[i_l, i_r - 1], S_2[j, m]), \\ \texttt{maxaps}(S_1[i_l + 1, i_r], S_2[j, m]) \end{array} \right\}$
       end for
    end for
    /* (2) Processing, in particular, those parts of $S_1$
     * which are outside all arcs */
    if ($\texttt{maxaps}(S_1[1, n], S_2[1, m]) = m$)
       then print '$S_2$ is an aps of $S_1$';
       else print '$S_2$ is not an aps of $S_1$';
    end if
```

Fig. 3. Outline in pseudo-code of the algorithm that solves APS(NESTED, PLAIN)

arc-preserving property is maintained: Since there are no arcs in S_2 we can match for an arc $(i_l, i_r) \in A_1$ either only its left endpoint or only its right endpoint to a base in S_2 (or none of them) but not both.
To determine the time needed to compute all entries in table T we, firstly, consider the time one call of maxaps takes, and, then the running time of Algorithm `aps_np` in Fig. 3 itself (proofs are omitted):

Lemma 1 *A call of maxaps*$(S_1[i_1, i_2], S_2[j_1, j_2])$ *takes* $O(|I'_1|)$ *time if* $S_1[i_1] \in I'_1$ *and* $S_1[i_2] \in I'_1$*, where either* $I'_1 = I_1^{(i_l, i_r)}$ *for an arc* $(i_l, i_r) \in A_1$ *or* $I'_1 = I_1$. □

Theorem 1 APS(NESTED, PLAIN) *can be solved in* $O(nm)$ *time.* □

4 APS(NESTED,CHAIN)

An instance of APS (NESTED, CHAIN) consists of an arc-annotated sequence S_1 with nested arc structure and a pattern S_2 with chain arc structure. As in Section 3, we use a dynamic programming table T of size $|A_1|m$.
We define, for $(j_l, j_r) \in A_2$, a *matching arc set* $\mathrm{MA}^{(j_l, j_r)}$ of all innermost matching arcs $(i_l, i_r) \in A_1$. Observe that no two arcs in $\mathrm{MA}^{(j_l, j_r)}$ are nested, i.e., for $(i_l^1, i_r^1), (i_l^2, i_r^2) \in \mathrm{MA}^{(j_l, j_r)}$, we have either $i_l^1 < i_r^1 < i_l^2 < i_r^2$ or $i_l^2 < i_r^2 < i_l^1 < i_r^1$.
It is decisive for our algorithm that, if several arc matches for an arc in A_2 are possible, we choose an innermost arc match: If S_2 is an aps of S_1 then S_2 can be matched to S_1 by assigning innermost arc matches to all arcs in A_2.
We start with a brief overview on our algorithm which, in essence, consists of two stages. In the first stage, we compute those table entries $T(i_l, j)$, where (i_l, i_r) is an arc in A_1 and $S_2[j]$ is a base inside an arc from A_2 or an endpoint of such an arc, analogously as in Section 3: we process the arcs in A_1 in increasing

Computation of maxaps_nc$(S_1[i_1, i_2], S_2[j_1, j_2])$:

- If $i_1 > i_2$ or $j_1 > j_2$ then maxaps_nc$(S_1[i_1, i_2], S_2[j_1, j_2]) := j_1 - 1$.
- If $i_1 = i_2$ then maxaps_nc$(S_1[i_1, i_1], S_2[j_1, j_2]) :=$
  - j_1, if $S_1[i_1] = S_2[j_1]$ and $S_2[j_1]$ is not an endpoint;
  - $j_1 - 1$, otherwise.
- If $i_1 < i_2$ and $j_1 = j_2$ then maxaps_nc$(S_1[i_1, i_2], S_2[j_1, j_1]) :=$
  - j_1, if $S_1[i_1] = S_2[j_1]$ and $S_2[j_1]$ is not an endpoint;
  - maxaps_nc$(S_1[i_1 + 1, i_2], S_2[j_1, j_1])$, if $S_1[i_1] \neq S_2[j_1]$ and $S_2[j_1]$ is not an endpoint;
  - $j_1 - 1$, if $S_2[j_1]$ is an endpoint.
- If $i_1 < i_2$ and $j_1 < j_2$ and neither $S_1[i_1]$ nor $S_2[j_1]$ is an endpoint then maxaps_nc$(S_1[i_1, i_2], S_2[j_1, j_2]) :=$
  - maxaps_nc$(S_1[i_1 + 1, i_2], S_2[j_1 + 1, i_2])$, if $S_1[i_1] = S_2[j_1]$;
  - maxaps_nc$(S_1[i_1 + 1, i_2], S_2[j_1, i_2])$, otherwise.
- If $i_1 < i_2$ and $j_1 < j_2$ and $S_2[j_1]$ is an endpoint of an arc but $S_1[i_1]$ is not then maxaps_nc$(S_1[i_1, i_2], S_2[j_1, j_2]) :=$ maxaps_nc$(S_1[i_1 + 1, i_2], S_2[j_1, j_2])$.
- If $i_1 < i_2$ and $j_1 < j_2$ and $S_1[i_1]$ is the left endpoint of an arc (i_l, i_r) then maxaps_nc$(S_1[i_1, i_2], S_2[j_1, j_2]) :=$ maxaps_nc$(S_1[i_r + 1, i_2], S_2[T[i_l, j_1] + 1, j_2])$.

Fig. 4. Recursive definition of maxaps_*nc*

order of their right endpoints. For every arc $(j_l, j_r) \in A_2$, we can, in the same way, also determine its innermost matching arcs $\mathrm{MA}^{(j_l, j_r)}$. In the second stage, we compute the table entries for bases in S_2 which are not inside an arc, using the function maxaps_nc which will be introduced in this section. Table T then is complete and we can compute maxaps_nc$(S_1[1, n], S_1[1, m])$ to decide whether S_2 is an arc-preserving subsequence of S_1.

The new function maxaps_nc extends the function maxaps by additionally taking the arc matching possibilities for arcs in A_2 into account: When processing those parts of S_2 which are not inside an arc, it allows to make use of the precomputed results in table T; thus, it treats the arcs of S_2 like single bases and skips them by matching them to an innermost arc match. The recursive definition of function maxaps_nc is given in Fig. 4. The two stages of the algorithm to solve APS(NESTED, CHAIN) are sketched in the following:

Stage 1: For every arc $(j_l, j_r) \in A_2$, we, firstly, compute $T(i_l, j)$, where $(i_l, i_r) \in A_1$ and $j_l < j < j_r$, i.e., j is inside (j_l, j_r):

$$T(i_l, j) := \max \left\{ \begin{array}{l} \text{maxaps}(S_1[i_l, i_r - 1], S_2[j, j_r - 1]), \\ \text{maxaps}(S_1[i_l + 1, i_r], S_2[j, j_r - 1]) \end{array} \right\}$$

Secondly, we compute $T(i_l, j_l)$, corresponding to the endpoints of the arc (j_l, j_r): If (i_l, i_r) is an innermost matching arc for (j_l, j_r), then we set $T(i_l, j_l) := j_r$, otherwise we reject the possibility of matching (i_l, i_r) with (j_l, j_r). By computing $T(i_l, j_l)$ in this way, we prefer the *innermost* arc matches in $\mathrm{MA}^{(j_l, j_r)}$ to other arc matches that would be possible. The test whether $(i_l, i_r) \in \mathrm{MA}^{(j_l, j_r)}$ is done as follows: We have an arc match if maxaps$(S_1[i_l{+}1, i_r{-}1], S_2[j_l{+}1, j_r{-}1]) = j_r{-}1$,

$S_1[i_l] = S_2[j_l]$, and $S_1[i_r] = S_2[j_r]$. To decide whether it is an *innermost* arc match, we recall that we process the arcs in A_1 in increasing order by their right endpoints. Therefore, we simply keep track of the previously found innermost arc match. If there was none so far or the match involved an arc $(i'_l, i'_r) \in A_1$ left of (i_l, i_r), i.e, $i'_l < i'_r < i_l < i_r$, then (i_l, i_r) is an innermost arc match.
Stage 2: After we have all matching possibilities for the arcs in A_2, we can now complete table T by processing those bases in S_2, which are outside all arcs, using function maxaps_nc which guarantees that *every* arc in S_2 has a matching arc in S_1. In the final step of our algorithm, if maxaps_nc$(S_1[1,n], S_2[1,m])$ returns m then S_2 is an arc-preserving subsequence of S_1. In summary, this yields the following result.

Theorem 2 APS(NESTED, CHAIN) *can be solved in* $O(nm)$ *time.* □

5 APS(NESTED,NESTED)

The basic idea how the algorithm for APS(NESTED,NESTED) builds on the algorithms in Sections 3 and 4 is as follows. We can process the bases inside of the innermost arcs in A_2 in the same way as in the algorithm for APS(NESTED,PLAIN). When processing an arc which is not innermost, i.e., there are arcs inside of it, we observe that the arcs which are directly inside this arc form a chain structure. Moreover, when processing the arcs in the order of their right endpoints, they are already processed at this point. Therefore, we can process these arcs in the same way as in the first stage of the algorithm for APS(NESTED,CHAIN).
The algorithm solving APS(NESTED,NESTED) is outlined in Fig. 5. In contrast to the algorithm for APS(NESTED, CHAIN), the first stage is extended to process all arcs in the nested arc structure of A_2: To fill the dynamic programming table as already used in the previous sections, we process the arcs in A_2 from inner to outer arcs and process, for every arc in A_2, the arcs in A_1 from inner to outer arcs. The second stage is, then, the same as in the algorithm for APS(NESTED, CHAIN): We complete the table for the bases in S_2 that are outside all arcs and, finally, we compute maxaps_nc$(S_1[1,n], S_2[1,m])$. If it returns m, then S_2 is an aps of S_1.

Theorem 3 APS(NESTED, NESTED) *can be solved in* $O(nm)$ *time.* □

6 AST(UNLIMITED,NESTED)

We use S_1 to denote the sequence with unlimited arc structure and, since, in contrast to the previous sections, there can be up to $O(|S_1|^2)$ many arcs, we use n to denote $|A_1|$. We use S_2 to denote the pattern sequence with nested arc structure and $m := |S_2|$.
In contrast to Section 5, the problem is, on the one hand, more general since S_1 can have unlimited arc structure. This makes it impossible to inspect the inside of the arcs by functions like maxaps (Section 3) or maxaps_nc (Section 4) which heavily relied on the nested arc structure of S_1. On the other hand, we

```
**Procedure** aps_nn
**Input**: Sequences $S_1$ and $S_2$ both with nested arc structures;
**Global variable**: **Array** of **int** $T[n][m]$;

  /************************* Stage 1 ******************************/
  for each $(j_l, j_r) \in A_2$ (ordered by their right endpoints) do
      for each $j \in I_2^{(j_l,j_r)}$ do
          for each $(i_l, i_r) \in A_1$ (ordered by their right endpoints) do
              if $(j_l, j_r)$ is an innermost arc then
                  $T(i_l, j) := \max \begin{Bmatrix} \texttt{maxaps}(S_1[i_l, i_r - 1], S_2[j, j_r - 1]), \\ \texttt{maxaps}(S_1[i_l + 1, i_r], S_2[j, j_r - 1]) \end{Bmatrix}$
              else
                  $T(i_l, j) := \max \begin{Bmatrix} \texttt{maxaps\_nc}(S_1[i_l, i_r - 1], S_2[j, j_r - 1]), \\ \texttt{maxaps\_nc}(S_1[i_l + 1, i_r], S_2[j, j_r - 1]) \end{Bmatrix}$
              end if
          end for
      end for
      for each $(i_l, i_r) \in A_1$ (ordered by their right endpoints) do

          $T(i_l, j_l) := \begin{cases} j_r & \text{if } (i_l, i_r) \in \texttt{MA}^{(j_l,j_r)}, \\ \texttt{maxaps\_nc}(S_1[i_l + 1, i_r], S_2[j_l, m]) & \text{otherwise.} \end{cases}$

      end for
  end for
  /************************* Stage 2 ******************************/
  for each $j \in I_2$ such that $S_2[j]$ is not an endpoint do
      for each $(i_l, i_r) \in A_1$ (ordered by their right endpoints) do
          $T(i_l, j) := \max \begin{Bmatrix} \texttt{maxaps\_nc}(S_1[i_l + 1, i_r], S_2[j, m]), \\ \texttt{maxaps\_nc}(S_1[i_l, i_r - 1], S_2[j, m]) \end{Bmatrix}$
      end for
  end for
  if $(\texttt{maxaps\_nc}(S_1[1, n], S_2[1, m]) = m)$
      then print '$S_2$ is an aps of $S_1$';
      else print '$S_2$ is not an aps of $S_1$';
  end if
```

Fig. 5. Outline in pseudo-code of the algorithm that solves APS(NESTED, NESTED)

have the additional restriction that every base is endpoint of at least one arc. This makes "partial" arc matches impossible, where we match only one but not the other endpoint of an arc in A_1 with a base in S_2 which is not an endpoint; this scenario constituted much of the computational difficulty of the problems in the previous sections. In the following, we outline how to adapt our dynamic programming techniques from the previous sections to the new problem.

Again, we build a dynamic programming table called T and compute its entries by processing the arcs of a nested arc structure, this time the one of S_2, in the order of their right endpoints. In contrast to the previous sections, only S_2 has a nested arc structure; therefore, we change the meaning of the entries in T. Now, table T has entries for every position in S_1 and every arc in A_2; we refer to the table entry corresponding to $1 \leq i \leq |S_1|$ and arc $(j_l, j_r) \in A_2$ by $T(i, j_r)$.

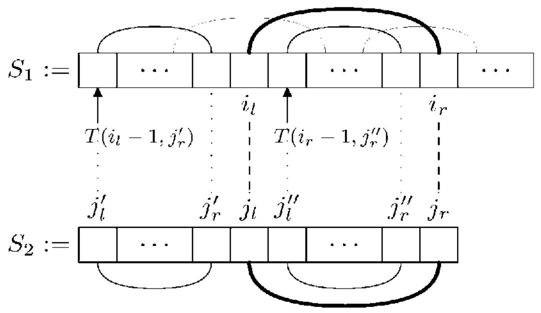


Fig. 6. Example illustrating how we determine whether $(j_l, j_r) \in A_2$ can be matched with $(i_l, i_r) \in A_1$ (match is indicated by the dashed lines) such that $S_2[\text{left}(j_r), j_r]$ is an ast of $S_1[1, i_r]$. Here, $S_2[\text{left}(j_r), j_r]$ *is* an ast of $S_1[1, i_r]$: Note that $\text{left}(j_r) = j_l'$ and (1) since $i_l < T(i_r - 1, j_r'')$ we can match $S_2[j_l'', j_r'']$ with $S_1[T(i_r - 1, j_r''), i_r - 1]$, and (2) since $T(i_l - 1, j_r') \neq -1$, we can match $S_2[j_l', j_r']$ with $S_1[T(i_l - 1, j_r'), i_l - 1]$ (these matches are indicated by the dotted lines)

To specify the entries of T, we need two additional definitions. Given j, $1 \leq j \leq m$, we use $\text{left}(j)$ to denote the minimum index $j' \leq j$ such that each of $S_2[j'], S_2[j'+1], \ldots, S_2[j]$ is endpoint of an arc $(j_l, j_r) \in A_2$ with $j_r \leq j$. Due to the order of processing the arcs in A_2, $S_2[\text{left}(j), j]$ is, intuitively speaking, the largest substring ending in $S_2[j]$ in which all bases are or have been examined when processing $S_2[j]$. Given $1 \leq j_1 < j_2 \leq m$ and $1 \leq i \leq |S_1|$, the *best match* of $S_2[j_1, j_2]$ at $S_1[i]$ denotes the maximum index i', $1 \leq i' \leq i$, such that $S_2[j_1, j_2]$ is an ast of $S_1[i', i]$ (and -1 if no such index exists). Now, we define the meaning of a T entry in a "dual" way compared to the previous sections: There, $T(i_l, j)$ corresponded to some arc $(i_l, i_r) \in A_1$ and $1 \leq j \leq m$ and it specified the *largest* possible substring *starting* in $S_2[j]$ which is an aps of $S_1[i_l, i_r]$. Here, in contrast, $T(i, j_r)$ corresponds to $1 \leq i \leq |S_1|$ and an arc $(j_l, j_r) \in A_2$ and it specifies the *smallest* possible substring which *ends* in $S_1[i]$ and of which $S_2[\text{left}(j_l), j_r]$ is an ast, i.e., it contains the best match of $S_2[\text{left}(j_l), j_r]$ at $S_1[i]$.
The decisive idea of the algorithm is to compute, for every arc $(j_l, j_r) \in A_2$ and every $1 \leq i \leq |S_1|$, the best match of $S_2[\text{left}(j_r), j_r]$ at $S_1[i]$. This value is stored in table entry $T(i, j_r)$. If, after all entries are computed, we have $T(|S_1|, m) \geq 1$ then S_2 is an ast of S_1. To compute the table entries, we process the arcs in A_2 by the order of their right endpoints. For each $(j_l, j_r) \in A_2$, we compute entries $T(i, j_r)$, for all $1 \leq i \leq |S_1|$, as follows: We loop through $i = 1, \ldots, |S_1|$, and process, for every i, all arcs ending at $S_1[i]$, $i_r = i$. For each of these arcs $(i_l, i_r) \in A_1$, we, firstly, set $i'' := i_l$ if $S_2[j_l, j_r]$ is an ast of $S_1[i_l, i_r]$ and i_l is larger than the current i'', and, secondly, we compute, the maximum index i', $1 \leq i' \leq i$, such that $S_2[\text{left}(j_r), j_l - 1]$ is an ast of $S_1[i', i'' - 1]$. How to compute i' is explained more precisely further below for an example situation. Now, during the loop $i = 1, \ldots, |S_1|$, we simply keep track of the currently maximum i'' and the currently maximum i' found so far. The currently best i' is stored in $T(i, j_r)$ after all arcs ending in $S_1[i]$ have been processed.

```
Procedure ast_un
Input: Sequence $S_1$ with unlimited arc structure and sequence $S_2$
       with nested arc structure, over unary alphabet, in both
       sequences every base being endpoint of at least one arc;
Global variable: Array of int $T[|S_1|][m]$;

   for each $(j_l, j_r) \in A_2$ (ordered by their right endpoints) do
      $i' := -1$; /* currently best match for $S_2$[left($j_r$), $j_r$] */
      $i'' := -1$; /* currently best match for $S_2[j_l, j_r]$ */
      for $i = 1$ to $|S_1|$ do
         for each $(i_l, i_r) \in A_1$ such that $i = i_r$ do
            /* (1) Compute the best match $i''$ such that
             * $S_2[j_l, j_r]$ is ast of $S_1[i'', i]$. */
            if $(j_r - j_l = 1)$ then /* $(j_l, j_r)$ is innermost arc */
               $i'' :=$ max$(i_l, i'')$;
            else /* there is an arc $(j_l', j_r - 1)$ */
               if $(i_l < T(i-1, j_r - 1))$ then $i'' :=$ max$(i_l, i'')$;
            end if;

            /* (2) Compute the best match $i'$ such that
             * $S_2$[left($j_r$), $j_r$] is ast of $S_1[i', i]$. */
            if $S_2[j_l - 1]$ is right endpoint of an arc then
               $i' :=$ max$(i', T(i'' - 1, j_l - 1))$;   /* T(-1,j):=-1 for all $j$ */
            else
               $i' :=$ max$(i', i'')$;
            end if;
         end for;
         $T(i, j_r) := i'$;
      end for;
   end for;
   if $(T(|S_1|, m) \neq -1)$
      then print '$S_2$ is arc substructure of $S_1$.';
      else print '$S_2$ is not arc substructure of $S_1$.';
   end if;
```

Fig. 7. Outline in pseudo-code of the algorithm that solves AST(UNLIMITED, NESTED)

The crucial point above is to determine, given $(i_l, i_r) \in A_1$ and $(j_l, j_r) \in A_2$, the maximum i' such that $S_2[\text{left}(j_r), j_r]$ is an ast of $S_1[i', i_r]$ while (i_l, i_r) is matched with (j_l, j_r). We explain how this question is divided into two separate parts using the example shown in Fig. 6:

Part 1: The inside of arc (j_l, j_r). We test whether $S_2[j_l + 1, j_r - 1]$ is an ast of $S_1[i_l + 1, i_r - 1]$ (otherwise, we cannot match (j_l, j_r) with (i_l, i_r)). In Fig. 6, $S_2[j_r - 1]$ is right endpoint of an arc $(j_l'', j_r'') \in A_2$. The table entries $T(i, j_r - 1)$, for all $1 \le i \le |S_1|$, have already been computed since $j_r - 1 = j_r'' < j_r$. Then, $S_2[j_l + 1, j_r - 1]$ is an ast of $S_1[i_l + 1, i_r - 1]$ iff $i_l < T(i_r - 1, j_r'')$.

Part 2: Left of arc (j_l, j_r). We compute the best match i', $1 \le i' \le i_l$, if existent, of $S_2[\text{left}(j_r), j_l - 1]$ at $S_1[i_l - 1]$ (only if such an $i' \ge 1$ exists then $S_2[\text{left}(j_r), j_r]$ is an ast of $S_1[1, i_r]$ while matching (i_l, i_r) with (j_l, j_r)). In Fig. 6,

$S_2[j_l - 1]$ is a right endpoint of an arc $(j'_l, j'_r) \in A_2$. The table entries $T(i, j_l - 1)$, for all $1 \leq i \leq |S_1|$, have already been computed since $j_l - 1 = j'_r < j_r$. Then, the best i' to be determined can be found in $T(i_l - 1, j'_r)$: If $T(i_l - 1, j'_r) \neq -1$, then $T(i_l - 1, j'_r)$ contains the best match of $S_2[j'_l, j'_r]$ at $S_1[i_l - 1]$. If, however, $T(i_l - 1, j'_r) = -1$ then $S_2[j'_l, j'_r]$ is not an ast of $S_1[1, i_l - 1]$ and no i' as we search for exists.
Summarizing, if Part 1 is answered positively, i.e., $S_2[j_l + 1, j_r - 1]$ is an ast of $S_1[i_l + 1, i_r - 1]$, and if we find an $i' \geq 1$ in Part 2, i.e., $S_2[\text{left}(j_r), j_l - 1]$ is an ast of $S_1[i', i_l - 1]$ for $i' \geq 1$ then this i' is the maximum i' such that $S_2[\text{left}(j_r), j_r]$ is an ast of $S_1[i', i_r]$ while (i_l, i_r) is matched with (j_l, j_r). The algorithm computing the table entries of T in this way is outlined in Fig. 7. Regarding the running time, note that, for every arc in A_2, we inspect every arc in A_1 once.

Theorem 4 AST(UNLIMITED, NESTED) *can be solved in time* $O(nm)$. □

For an easier presentation we focused here on the case of unary alphabet and the restriction that the endpoints of an arc are single bases. It is conceivable that the algorithm can be extended to the case of non-unary alphabet as well as to the problem as stated by Vialette [9] (in this case the endpoints of arcs are given by intervals), showing that his PATTERN MATCHING OVER 2-INTERVAL SET restricted to $\{<, \sqsubset\}$ structured patterns is solvable in $O(n \log n + nm)$ time (details omitted).

References

1. J. Alber, J. Gramm, J. Guo, and R. Niedermeier. Towards optimally solving the LONGEST COMMON SUBSEQUENCE problem for sequences with nested arc annotations in linear time. In *Proc. of 13th CPM*, number 2373 in LNCS, pages 99–114, 2002. Springer.
2. V. Bafna, S. Muthukrishnan, and R. Ravi. Computing similarity between RNA strings. In *Proc. of 6th CPM*, number 937 in LNCS, pages 1–16, 1995. Springer.
3. N. El-Mabrouk and M. Raffinot. Approximate matching of secondary structures. In *Proc. of 6th ACM RECOMB*, pages 156–164, 2002. ACM Press.
4. P. A. Evans. Finding common subsequences with arcs and pseudoknots. In *Proc. of 10th CPM*, number 1645 in LNCS, pages 270–280, 1999. Springer.
5. T. Jiang, G.-H. Lin, B. Ma, and K. Zhang. The longest common subsequence problem for arc-annotated sequences. In *Proc. of 11th CPM*, number 1848 in LNCS, pages 154–165, 2000. Springer. To appear in *Journal of Discrete Algorithms.*
6. T. Jiang, G. Lin, B. Ma, and K. Zhang. A general edit distance between RNA structures. *Journal of Computational Biology* 9(2): 371–388, 2002.
7. G.-H. Lin, Z.-Z. Chen, T. Jiang, and J. Wen. The longest common subsequence problem for sequences with nested arc annotations. In *Proc. of 28th ICALP*, number 2076 in LNCS, pages 444–455, 2001. Springer. To appear in *Journal of Computer and System Sciences.*
8. P. Kilpeläinen and H. Mannila. Ordered and unordered tree inclusion. *SIAM Journal on Computing* 24(2): 340–356, 1995.
9. S. Vialette. Pattern matching problems over 2-interval sets. In *Proc. of 13th CPM*, number 2373 in LNCS, pages 53–63, 2002. Springer.

Knowledge over Dense Flows of Time (from a Hybrid Point of View)

Bernhard Heinemann

Fachbereich Informatik, FernUniversität Hagen
PO Box 940, D–58084 Hagen, Germany
Bernhard.Heinemann@fernuni-hagen.de
Phone: + 49 2331 987 2714, Fax: + 49 2331 987 319

Abstract. This paper is about an application of hybrid logic to the following problem with reasoning about knowledge: how to axiomatize spaces of knowledge states over dense flows of time? We provide an answer to this question below, proving a corresponding completeness theorem. We will make essential use of the fact that the hybrid logical language is particularly suited to express, in the modal sense, first–order properties of frames.

Keywords: modal logic of subset spaces, reasoning about knowledge and time, hybridization

1 Introduction

The idea of *knowledge* has proved to be a useful tool for analysing the behaviour of agents involved in a distributed environment. Originating from philosophical logic, cf [11], the formal description of knowledge by means of modal logic has been put in concrete terms by computer scientists since the eighties. The textbooks [5] and [13] give a good account of the state of the art in the various areas of application related to informatics.

Some years ago, an interesting connection between knowledge and *topology* was discovered and exploited for both the logic of knowledge and topological reasoning. The first paper regarding this is [14], and the fundamental one [4] (the journal version of [14]). Let us refer to the system emerging from these papers as to the *modal logic of subset spaces (MLSS),* for convenience. (The suggestive term 'topologic' is usual, too.) Further developments of *MLSS* were provided after that, for example, by [6], [8], [10], and [15]. In particular, systems were studied in these papers which are sensitive to the respective semantics underlying any knowledge acquisition procedure, viz *linear* or *branching time.*

The language of *MLSS* enables one to specify certain properties of 'topological' spaces $(X, \mathcal{O})$ where X is a non–empty set (of states) and $\mathcal{O}$ a set of subsets of X, the distinguished *system of opens* or *neighbourhoods* (representing potential knowledge states). *MLSS*–formulas are evaluated at *neighbourhood situations* consisting of a state and a neighbourhood thereof. In case of a single agent there are two modal connectives contained in the language, K and $\Box$, quantifying across states *close* to the actual one and sets from $\mathcal{O}$ *refining* the current open, respectively. One can think of K as a *knowledge operator* and $\Box$ as an *effort operator* assigned to the agent (being busy with acquiring knowledge, that is, topologically, shrinking neighbourhoods and approximating objects thus). Both modalities interact according to the structure of the space under discussion.

Though approriate to several tasks in formal reasoning about knowledge *MLSS* has some shortcomings. First, these are the usual ones of every modal system; that is, in particular, distinguished states cannot be addressed directly. (This is sometimes felt as a deficiency, even in contexts where on the whole modal logic is applied successfully; cf [1].) Second, it has turned out that some naturally arising models of knowledge cannot, or at least not obviously, be treated by means of *MLSS*. A prominent example for the latter is *knowledge evolving over dense time.* Dense (or even continous) flows of time might be the adequate temporal basis for certain high–level specifications of the time–dependent behaviour of knowledge. So, one should have corresponding modelling tools at one's disposal.

This paper rectifies the lack of each of the properties just mentioned. Dense subset spaces are viewed as models of an extended, so–called *hybrid* language here; cf [2], Section 7.3, or [1]. As far as shrinking of sets is concerned the present paper is complementary to our contribution to FST&TCS'99 in a sense; cf [7]. However, applying hybrid methods goes far beyond that purely modal context in many respects, as will become apparent later on.

How were we led to this approach? Our starting point was a discussion on the treatment of dense subset spaces, at FST&TCS'99[1]. Assuming one claims a completeness result for the class of these structures, it is apparently not possible then to succeed along the lines of Moss and Parikh's proof for general subset spaces (where the step–by–step method was applied; cf [4] and [2], 4.6). Extended modal techniques like infinite axiomatizations (see [15]) or rules for the undefinable (like Gabbay's irreflexivity rule; cf [2], 4.7) cannot obviously be utilized for that purpose either. At this point hybrid logic offers new and powerful abilities: *naming neighbourhood situations explicitly* not only extends the means of expression (which is useful in practice as it throws a bridge to implementation–oriented tableaux procedures; cf [1]), but also helps us with our basic tasks. Note that, in the context of knowledge, such names distinguish *pointed knowledge states.*

What particulars is the current paper about? First, we define precisely the hybrid language of subset spaces indicated above, extending the modal one by nominals as names of neighbourhood situations and corresponding jump operators. Afterwards we briefly revisit usual hybrid logic and argue that the completeness proof contained in [2], Section 7.3, can be transferred to the bimodal case almost literally, which is relevant to the framework of this paper. Then, we give an axiomatization of the hybrid logic of arbitrary subset spaces, of which we prove 'canonical' completeness. This result is the main issue of the paper. Almost as a by–product of the general theory we get the desired completeness re-

[1] I am grateful to R. Ramanujam for corresponding questions there.

sult for dense subset spaces. Finally, we discuss some demanding open problems concerning the new logic.

2 A Hybrid Language for Subset Spaces

In this section we define the syntax and semantics of the language underlying our investigations. In essence, we provide *MLSS* with a certain 'naming–and–referring–to' mechanism. To be more precise, we add names of neighbourhood situations and assign a connective to every name by means of which one is able to evaluate a formula at the denotation of this name.

Let PROP and NNS be two disjoint sets of symbols called *propositional variables* and *names of neighbourhood situations,* respectively. We use p, q, ... to denote typical elements of PROP, and $\mathfrak{n}, \mathfrak{m}$, ... for the same purpose in case of neighbourhood situations. Moreover, we assume that there is a distinguished name $\mathfrak{n}_0 \in$ NNS, because of technical reasons which will become clear later on. We define the set WFF of *well–formed formulas* of a hybrid bimodal language of subset spaces over PROP and NNS by the generative rule

$$\alpha \ ::= \ p \mid \mathfrak{n} \mid \neg\alpha \mid \beta \wedge \alpha \mid K\alpha \mid \Box\alpha \mid @_{\mathfrak{n}}\alpha.$$

In particular, we designate formulas by lower case Greek letters. The missing boolean connectives $\top, \bot, \vee, \rightarrow, \leftrightarrow$ are treated as abbreviations, as needed. The duals of the one–place modalities K and $\Box$ are written L and $\Diamond$, respectively. Finally, denoting formulas we use the common priority rules.

Hybrid logic deals with *nominals* from a set NOM of names of states, and *satisfaction operators* $@_i$ where $i \in$ NOM, among other things. We replace NOM by NNS, and $@_i$ by $@_{\mathfrak{n}}$ correspondingly. There is no need to distinguish between $@_{\mathfrak{n}}$ and its dual because $@_{\mathfrak{n}}$ turns out to be self–dual, as in usual hybrid logic. It should be noted that we do not consider other hybrid binders like the $\downarrow$–operator in the technical part of this paper.

We will next give meaning to formulas. For this purpose we have to define the relevant structures first. These are essentially the same domains as of the modal logic of subset spaces, but the point–dependent only valuation of the propositional variables is to be extended to NNS suitably. Let $\mathcal{P}(A)$ designate the powerset of a given set A in the following.

Definition 1 (Subset frames; hybrid subset spaces).

1. *Let X be a non–empty set and $\mathcal{O} \subseteq \mathcal{P}(X)$ a system of non–empty subsets of X such that $X \in \mathcal{O}$. Then the pair $(X, \mathcal{O})$ is called a* subset frame.
2. *Let $(X, \mathcal{O})$ be a subset frame. The set of* neighbourhood situations *of $(X, \mathcal{O})$ is $\mathcal{N} := \{x, U \mid (x, U) \in X \times \mathcal{O}$ and $x \in U\}$. (Neighbourhood situations are written without brackets.)*
3. *A* hybrid subset space *is a triple $(X, \mathcal{O}, V)$, where $(X, \mathcal{O})$ is a subset frame and V a* hybrid valuation*, i.e., a mapping*

$$V : \text{PROP} \cup \text{NNS} \longrightarrow \mathcal{P}(X) \cup \mathcal{N}$$

such that

(a) $V(p) \subseteq \mathcal{P}(X)$ *for all* $p \in \mathrm{PROP}$,
(b) $V(\mathfrak{n}) \in \mathcal{N}$ *for all* $\mathfrak{n} \in \mathrm{NNS}$, *and*
(c) $V(\mathfrak{n}_0) = x, X$ *for some* $x \in X$.
$\mathcal{M} := (X, \mathcal{O}, V)$ *is said to* be based on $(X, \mathcal{O})$.

Two things must be mentioned here. First, the valuation of the propositional variables goes to *sets*. As we will define the semantics of formulas with respect to *neighbourhood situations* immediately, this forces a special axiom only relevant to propositional variables. In particular, the set of validities cannot be closed under substitution therefore. Second, the special name $\mathfrak{n}_0$ really lies *between* nominals and constants: it is true that the denotation of $\mathfrak{n}_0$ consists of exactly one neighbourhood situation, but only those having X as its set component are admissible.

For a given hybrid subset space we now define the satisfaction relation $\models$ between neighbourhood situations of the underlying frame and formulas in WFF. We omit the obvious clauses for the boolean connectives.

Definition 2 (Satisfaction and validity). *Given a subset frame* $\mathcal{S} := (X, \mathcal{O})$, *a hybrid subset space* $\mathcal{M} := (X, \mathcal{O}, V)$ *based on it, and a neighbourhood situation* x, U *of* $\mathcal{S}$, *then*

$$\begin{array}{lcl}
x, U \models_{\mathcal{M}} p & :\Longleftrightarrow & x \in V(p) \\
x, U \models_{\mathcal{M}} \mathfrak{n} & :\Longleftrightarrow & V(\mathfrak{n}) = x, U \\
x, U \models_{\mathcal{M}} K\alpha & :\Longleftrightarrow & y, U \models_{\mathcal{M}} \alpha \text{ for all } y \in U \\
x, U \models_{\mathcal{M}} \Box\alpha & :\Longleftrightarrow & x, U' \models_{\mathcal{M}} \alpha \text{ for all } U' \in \mathcal{O} \text{ such that } x \in U' \subset U \\
x, U \models_{\mathcal{M}} @_{\mathfrak{n}}\alpha & :\Longleftrightarrow & V(\mathfrak{n}) \models_{\mathcal{M}} \alpha,
\end{array}$$

for all $p \in \mathrm{PROP}$, $\mathfrak{n} \in \mathrm{NNS}$, *and* $\alpha \in \mathrm{WFF}$. *In case* $x, U \models_{\mathcal{M}} \alpha$ *is true we say that* α holds in $\mathcal{M}$ at *the neighbourhood situation* x, U. *The formula* α *is called* valid in $\mathcal{M}$, *iff it holds in* $\mathcal{M}$ *at every neighbourhood situation. Moreover, the notion of validity is extended to subset frames* $\mathcal{S}$ *by quantifying over all hybrid subset spaces based on* $\mathcal{S}$. *(Manners of writing:* $\mathcal{M} \models \alpha$ *and* $\mathcal{S} \models \alpha$, *respectively.)*

Note that the relation $\subset$ used in the clause for $\Box$ means *proper* set inclusion. This is due to the special classes of frames we will be interested in later on in the paper, viz *dense* and *linear* ones, respectively.

Definition 3 (Dense subset frames; linear subset frames). *A subset frame* $(X, \mathcal{O})$ *is called* dense, *iff for all* $U, U' \in \mathcal{O}$ *there exists* $U'' \in \mathcal{O}$ *such that* $U \subset U'' \subset U'$. *And* $(X, \mathcal{O})$ *is called* linear, *iff the relation* $\subset$ *is trichotomous on* $\mathcal{O}$ *(i.e., for all* $U, U' \in \mathcal{O}$ *it holds that* $U \subset U'$ *or* $U' \subset U$ *or* $U = U'$.*)*

As an example of valid formulas we state the hybrid correspondents of the basic properties of the proper set inclusion relation $\subset$, viz irreflexivity and transitivity, making the structure $(\mathcal{O}, \subset)$ a strict partial order. The straightforward proof of the following proposition is omitted.

Proposition 1 (Expressing properties of $\subset$**).** *Let* $\mathcal{S} = (X, \mathcal{O})$ *be a subset frame and* $\mathfrak{n} \in \mathrm{NNS}$ *a name of a neighbourhood situation. Then both* $\mathfrak{n} \to \neg\Diamond\mathfrak{n}$ *and* $\Diamond\Diamond\mathfrak{n} \to \Diamond\mathfrak{n}$ *are valid in* $\mathcal{S}$.

3 Revisiting Basic Hybrid Logic

We set our sights now on a hybrid logical system for subset spaces. For a start, we list the bulk of the Hilbert–style axiomatization of hybrid logic given in [2], p 438 ff, which originates from [3]. Then, that system is adapted slightly to the bimodal case. The corresponding completeness proof is sketched (or, rather, the original one is revisited and the minor modifications are indicated; cf [2], loc cit). Note that all axiom and rule schemata given in this section are sound on hybrid subset spaces.

1. $@_{\mathfrak{n}}(\alpha \to \beta) \to (@_{\mathfrak{n}}\alpha \to @_{\mathfrak{n}}\beta)$
2. $@_{\mathfrak{n}}\alpha \leftrightarrow \neg @_{\mathfrak{n}}\neg\alpha$
3. $\mathfrak{n} \wedge \alpha \to @_{\mathfrak{n}}\alpha$
4. $@_{\mathfrak{n}}\mathfrak{n}$
5. $@_{\mathfrak{n}}\mathfrak{m} \wedge @_{\mathfrak{m}}\alpha \to @_{\mathfrak{n}}\alpha$
6. $@_{\mathfrak{n}}\mathfrak{m} \leftrightarrow @_{\mathfrak{m}}\mathfrak{n}$
7. $@_{\mathfrak{m}}@_{\mathfrak{n}}\alpha \leftrightarrow @_{\mathfrak{n}}\alpha$,

where $\mathfrak{n}, \mathfrak{m} \in$ NNS and $\alpha, \beta \in$ WFF. We sum up next what can be achieved with the aid of these two blocks of axioms. Let Γ be a maximal consistent set of formulas and $\mathfrak{n}$ a name of a neighbourhood situation. One can derive from Γ the set $\Delta_{\mathfrak{n}} := \{\alpha \mid @_{\mathfrak{n}}\alpha \in \Gamma\} \subseteq$ WFF then, which represents a natural candidate for realizing formulas of the type $@_{\mathfrak{n}}\alpha$ contained in Γ. The properties of $\Delta_{\mathfrak{n}}$ needed for this purpose can in fact be proved with the aid of the above axioms and the familiar derivation rules *modus ponens* and *necessitation;* cf [2], 7.24. In particular, $\Delta_{\mathfrak{n}}$ is a uniquely determined maximal consistent set containing $\mathfrak{n}$. The axiom schema called (*back*) in [2] occurs twice here because we have to take into account both modalities, K and $\Box$ (viewed as K–modalities at the moment):

8. $L@_{\mathfrak{n}}\alpha \to @_{\mathfrak{n}}\alpha$
9. $\Diamond @_{\mathfrak{n}}\alpha \to @_{\mathfrak{n}}\alpha$

The same is true for the rule (PASTE) so that we have three additional rules:

$$\text{(NAME)}\ \frac{\mathfrak{m} \to \beta}{\beta} \qquad \text{(PASTE)}_K\ \frac{@_{\mathfrak{n}}L\mathfrak{m} \wedge @_{\mathfrak{m}}\alpha \to \beta}{@_{\mathfrak{n}}L\alpha \to \beta}$$

$$\text{(PASTE)}_\Box\ \frac{@_{\mathfrak{n}}\Diamond\mathfrak{m} \wedge @_{\mathfrak{m}}\alpha \to \beta}{@_{\mathfrak{n}}\Diamond\alpha \to \beta},$$

where $\alpha, \beta \in$ WFF, $\mathfrak{n}, \mathfrak{m} \in$ NNS, and $\mathfrak{m}$ is 'new' each time.

The axioms and rules stated so far enable one to perform the naming and pasting techniques of hybrid logic, respectively, in the present case as well; cf [2], p 441 ff. That is, we obtain a hybrid bimodal model $\mathcal{M}$ satisfying the following properties:

– $\mathcal{M}$ is yielded (in the sense of [2], Definition 7.26) by a maximal consistent set Γ containing the negation of a given non–derivable formula γ. In particular, every point of $\mathcal{M}$ is some $\Delta_{\mathfrak{n}}$, where $\mathfrak{n}$ is taken from some suitably extended set of names of neighbourhood situations. ('$\mathcal{M}$ is named'.)

- The modalities K and $\Box$ induce respective accessibility relations $\xrightarrow{L}$ and $\xrightarrow{\Diamond}$ on $\mathcal{M}$, for which the *Existence Lemma* holds; i.e., if s is a point containing $L\alpha$, then there exists a point t such that $s \xrightarrow{L} t$ and $\alpha \in t$ (and for $\Diamond\alpha$ and $\xrightarrow{\Diamond}$ correspondingly). ('$\mathcal{M}$ is pasted'.)
- The *Truth Lemma* holds for $\mathcal{M}$; in particular, $\mathcal{M}$ falsifies γ at Γ.

This easily gives us completeness. But we can get even more: axiomatizing special properties of the relations $\xrightarrow{L}$, $\xrightarrow{\Diamond}$, and the composition of both of them, by means of *pure* formulas, i.e., without using propositional variables, yields that the frame underlying the 'canonical' model $\mathcal{M}$ shares these properties; cf [2], Theorem 7.29. This fact will be used extensively in the subsequent sections.

4 General Subset Spaces

In this section we exploit the hybrid canonicity just mentioned for proving completeness of the logic of subset spaces with no restrictions imposed on the set of opens. The additional axioms required for this are listed below. For a start, we state the hybrid version of most of the usual modal axioms for subset spaces, cf [4]. (We leave out, in particular, the distribution schemata for K and $\Box$, which were implicitly used in the previous section already.)

10. $\mathfrak{n} \rightarrow L\mathfrak{n}$ 13. $(p \rightarrow \Box p) \wedge (\neg p \rightarrow \Box\neg p)$
11. $LL\mathfrak{n} \rightarrow L\mathfrak{n}$ 14. $\Diamond\Diamond\mathfrak{n} \rightarrow \Diamond\mathfrak{n}$
12. $L\mathfrak{n} \rightarrow KL\mathfrak{n}$ 15. $\Diamond L\mathfrak{n} \rightarrow L\Diamond\mathfrak{n}$,

where p varies across PROP and $\mathfrak{n}$ across NNS.

Some remarks are opportune here. First, the schemata 10 – 12 represent the hybrid axioms of knowledge, corresponding to the well–known modal S5. Second, Axiom 13 is *not* pure (and it is the only one having this property, apart from the distribution schemata). This axiom corresponds to evaluating propositional variables with respect to points only; cf Definition 2 and the remark subsequent to Definition 1. It turns out that this kind of non–purity does not cause any problems concerning hybrid completeness. Third, the axiom for reflexivity is missing here; this must clearly be the case because of the clause for $\Box\alpha$ in Definition 2. But also the axiom of irreflexivity (cf Proposition 1) does not occur. (It is actually not needed later on.) Finally, Axiom 15 connects the modal operators contained in our language with regard to *shrinking*. – The above list is supplemented by the following axiom having no counterpart in [4]:

16. $@_{\mathfrak{n}_0} L\Diamond\mathfrak{n}$,

where $\mathfrak{n} \in$ NNS. Axiom 16 corresponds to the fact that the whole space X always belongs to the opens, and is the largest one thus; cf Definition 1.1.

We continue working with the model $\mathcal{M}$ from the previous section. Because of purity, Axioms 10 – 12 and 14 – 15, respectively, give us right off the following basic properties required of the relations $\xrightarrow{L}$ and $\xrightarrow{\Diamond}$ later on:

- the relation $\xrightarrow{L}$ is an equivalence relation,
- the relation $\xrightarrow{\Diamond}$ is transitive, and
- the following *cross property* holds for $\xrightarrow{L}$ and $\xrightarrow{\Diamond}$: for all points s, t, u such that $s \xrightarrow{\Diamond} t \xrightarrow{L} u$, there exists a point v such that $s \xrightarrow{L} v \xrightarrow{\Diamond} u$.

For all points s of the model $\mathcal{M}$, let $[s]$ denote the $\xrightarrow{L}$–equivalence class of s. We say then that $[s]$ *precedes* $[t]$, and write $[s] \prec [t]$, iff there are $s' \in [s]$ and $t' \in [t]$ such that $s' \xrightarrow{\Diamond} t'$. With the aid of the *cross property* it is easy to prove that this relation of preceding is transitive. We are going to establish now six crucial properties of the relation $\prec$.

Definition 4 (Crucial properties of $\prec$). *We call the relation* $\prec$

1. exclusively induced by $\xrightarrow{\Diamond}$, *iff, whenever* $[s] \prec [t]$ *and* t' *is an* $\xrightarrow{\Diamond}$*–successor of* $s' \in [s]$ *contained in* $[t]$*, then* $s' \xrightarrow{L} t'$ *does not hold,*
2. functionally induced by $\xrightarrow{\Diamond}$, *iff, whenever* $[s] \prec [t]$*, then every* $s' \in [s]$ *has at most one* $\xrightarrow{\Diamond}$*–successor contained in* $[t]$,
3. injectively induced by $\xrightarrow{\Diamond}$, *iff, whenever* $[s] \prec [t]$*, then no two distinct points* $s', s'' \in [s]$ *have a common* $\xrightarrow{\Diamond}$*–successor contained in* $[t]$*, and*
4. faithfully induced by $\xrightarrow{\Diamond}$, *iff, whenever* $[s] \prec [t]$, $[s] \prec [u]$ *and not* $[t] \prec [u]$*, then there exists a point* $s' \in [s]$ *having an* $\xrightarrow{\Diamond}$*–successor contained in* $[u]$*, but no* $\xrightarrow{\Diamond}$*–successor contained in* $[t]$.
5. *Furthermore, the relation* $\prec$ *is said to* satisfy the factor property*, iff, whenever* $[s] \prec [t]$ *and* $[t] \prec [u]$*, then every* $s' \in [s]$ *having an* $\xrightarrow{\Diamond}$*–successor contained in* $[u]$ *also has an* $\xrightarrow{\Diamond}$*–successor contained in* $[t]$.
6. *Finally,* $\prec$ *is called* true*, iff, whenever* $[s] \prec [t]$*, then there exists* $s' \in [s]$ *having no* $\xrightarrow{\Diamond}$*–successor contained in* $[t]$.

It can easily be seen that the above properties are implied by the respective first–order conditions written down in the next proposition. Pure correspondents are additionally delivered there, which are sound on hybrid subset spaces.

Proposition 2 (Defining these properties). *Let* $\mathcal{F} := (W, \{R, S\})$ *be a bimodal Kripke frame, where* R *corresponds to the modality* K *and* S *to* $\Box$*. Then:*

1. $\mathcal{F} \models \Diamond\mathfrak{n} \to \neg L\mathfrak{n}$, *iff* $\forall\, w, v, u \in W : (w\, S\, v \Rightarrow \neg(w\, R\, v))$.
2. $\mathcal{F} \models \Diamond\mathfrak{n} \wedge \Diamond(\mathfrak{m} \wedge L\mathfrak{n}) \to \Diamond(\mathfrak{n} \wedge \mathfrak{m})$, *iff*
$$\forall\, w, v, u \in W : (w\, S\, v \wedge w\, S\, u\, R\, v \Rightarrow v = u).$$
3. $\mathcal{F} \models \neg\mathfrak{n} \wedge \Diamond\mathfrak{m} \to K(\mathfrak{n} \to \neg\Diamond\mathfrak{m})$, *iff*
$$\forall\, w, v, u \in W : (w\, S\, u \wedge v\, S\, u \wedge w\, R\, v \Rightarrow w = v).$$

4. $\mathcal{F} \models \Diamond(\mathfrak{n} \wedge \neg\Diamond L\mathfrak{m}) \wedge L\Diamond\mathfrak{m} \to L(\Diamond L\mathfrak{m} \wedge \neg\Diamond L\mathfrak{n})$, *iff*

$$\forall\, w,v,u,t \in W : (w\,S\,v \wedge w\,R\,u\,S\,t \wedge \forall\, x : \neg(v\,R\,x\,S\,t) \\ \Rightarrow \exists\, s,r : (w\,R\,s\,S\,r\,R\,t \wedge \forall\, y : \neg(s\,S\,y\,R\,v))).$$

5. $\mathcal{F} \models \Diamond\mathfrak{m} \wedge L\Diamond(\mathfrak{n} \wedge \Diamond\mathfrak{m}) \to \Diamond(L\mathfrak{n} \wedge \Diamond\mathfrak{m})$, *iff*

$$\forall\, w,v,u,t \in W : (w\,S\,v \wedge w\,R\,u\,S\,t\,S\,v \Rightarrow \exists\, s : (w\,S\,s\,S\,v \wedge s\,R\,t))\,.$$

6. $\mathcal{F} \models \Diamond\mathfrak{n} \to L\neg\Diamond L\mathfrak{n}$, *iff*

$$\forall\, w,v \in W : (w\,S\,v \Rightarrow \exists\, u : (w\,R\,u \wedge \forall\, t : \neg(u\,S\,t\,R\,v)))\,.$$

Proof. To exemplify the manner of proceeding we prove the right–to–left direction of (iii) and the left–to–right direction of (v). We use the terminology common in hybrid logic; cf [2], 7.3.

First, assume that we have $\forall\, w,v,u \in W : (w\,S\,u \wedge v\,S\,u \wedge w\,R\,v \Rightarrow w = v)$. Let $\mathcal{M}$ be an arbitrary hybrid model based on $\mathcal{F}$ and $w \in W$ a point such that $\mathcal{M}, w \models \neg\mathfrak{n} \wedge \Diamond\mathfrak{m}$. Let $u \in W$ be the denotation of $\mathfrak{m}$, i.e., $\mathcal{M}, u \models \mathfrak{m}$. Furthermore, let $v \in W$ be a point such that $w\,R\,v$ and $\mathcal{M}, v \models \mathfrak{n}$. Then we conclude $w \neq v$. It follows from the above first–oder condition that $v\,S\,u$ does not hold. Consequently, $\mathcal{M}, v \not\models \Diamond\mathfrak{m}$. This shows $\mathcal{F} \models \neg\mathfrak{n} \wedge \Diamond\mathfrak{m} \to K(\mathfrak{n} \to \neg\Diamond\mathfrak{m})$.

Now suppose that the condition stated in (v) is violated, i.e., $\exists\, w,v,u,t \in W : (w\,S\,v \wedge w\,R\,u\,S\,t\,S\,v \wedge \forall\, s : (w\,S\,s \Rightarrow ((s,v) \notin S \vee (s,t) \notin R)))$. Take a hybrid valuation on $\mathcal{F}$ satisfying $V(\mathfrak{n}) = \{t\}$ and $V(\mathfrak{m}) = \{v\}$. Let $\mathcal{M} := (\mathcal{F}, V)$. Then $\mathcal{M}, w \models \Diamond\mathfrak{m} \wedge L\Diamond(\mathfrak{n} \wedge \Diamond\mathfrak{m})$ and $\mathcal{M}, w \models \Box(K\neg\mathfrak{n} \vee \Box\neg\mathfrak{m})$, thus $\mathcal{F} \not\models \Diamond\mathfrak{m} \wedge L\Diamond(\mathfrak{n} \wedge \Diamond\mathfrak{m}) \to \Diamond(L\mathfrak{n} \wedge \Diamond\mathfrak{m})$.

Thus adding the formulas occurring in Proposition 2 to our system (as Axioms 17 – 22), a canonicity argument as above yields that the properties stated in Definition 4 are in fact valid for $\prec$.

Corollary 1 ($\prec$ meets Definition 4). *In the presence of all of the above axioms, the relation $\prec$ is exclusively, functionally, injectively, and faithfully induced by $\xrightarrow{\Diamond}$. Moreover, $\prec$ satisfies the factor property and is true.*

Let $\mathcal{M}$ be as above and γ a non–derivable formula. Corollary 1 puts us now in a position to define a hybrid subset space falsifying γ, in the following way. Let V be the distinguished hybrid valuation of $\mathcal{M}$ and $s_0 := V(\mathfrak{n}_0)$ the denotation of the special name $\mathfrak{n}_0 \in$ NNS. Then the carrier set of the desired hybrid subset space is defined as the $\xrightarrow{L}$–equivalence class of s_0 :

$$X := [s_0]\,.$$

Let I be the set of points t of $\mathcal{M}$ such that $[s_0]$ precedes $[t]$. Note that, given such a t, for every $s \in [s_0]$ there exists at most one $\xrightarrow{\Diamond}$–successor t_s of s contained in $[t]$; this is due to the 'functionality' assertion of Corollary 1. For all $t \in I$, an open U_t is defined by

$$U_t := \{s \in X \mid s \text{ has an } \xrightarrow{\Diamond}\text{– successor contained in } [t]\}.$$

Furthermore, $\mathcal{O} := \{U_t \mid t \in I\}$. Finally, we define tentatively a hybrid valuation $\widetilde{V}$ on the subset frame $(X, \mathcal{O})$ by

$$\begin{aligned} \widetilde{V}(p) &:= X \cap V(p) \quad \text{for all } p \in \text{PROP}, \text{ and} \\ \widetilde{V}(\mathfrak{n}) &:= s_{\mathfrak{n}}, U_{V(\mathfrak{n})} \quad \text{for all } \mathfrak{n} \in \text{NNS}, \end{aligned}$$

where $s_{\mathfrak{n}}$ is the $\xrightarrow{\Diamond}$–predecessor of $V(\mathfrak{n})$ contained in $[s_0]$. Note that $s_{\mathfrak{n}}$ is in fact uniquely determined because of the 'injectivity' assertion of Corollary 1, if a point like that exists at all (that is, if $[s_0]$ precedes $[V(\mathfrak{n})]$). And we actually get that $\widetilde{\mathcal{M}} := (X, \mathcal{O}, \widetilde{V})$ bears the right structure.

Proposition 3. *$\widetilde{\mathcal{M}}$ is a hybrid subset space such that*

1. *the relation $\prec$ is asymmetric, and*
2. *$[t] \prec [u]$ iff $U_u \subset U_t$, for all $t, u \in I$.*

Proof. We first have to show that $\widetilde{V}$ is well–defined. For this purpose note that $\mathcal{M}$ is generated by the denotation s_0 of $\mathfrak{n}_0$, due to Axiom 16; to be more precise, this axiom implies that for all points t of $\mathcal{M}$ it holds that

$$(s_0, t) \in \xrightarrow{L} \circ \xrightarrow{\Diamond}.$$

Consequently, the element $s_{\mathfrak{n}}$ occurring in the definition of $\widetilde{V}$ in fact exists because of the *cross property* (and is uniquely determined thus).

Now we turn to the asserted properties of $\prec$:

1. Suppose that $[t] \prec [u]$ and $[u] \prec [t]$. Then $[t] \prec [t]$ since $\prec$ is transitive. Thus, there exist $t', t'' \in [t]$ such that $t' \xrightarrow{\Diamond} t''$. But we clearly have $t' \xrightarrow{L} t''$. This contradicts the fact that $\prec$ is exclusively induced by $\xrightarrow{\Diamond}$. Hence $[u] \not\prec [t]$ if $[t] \prec [u]$, i.e., $\prec$ is asymmetric.
2. First assume that $[t] \prec [u]$. Let $s \in U_u$. Then there exists $u' \in [u]$ such that $s \xrightarrow{\Diamond} u'$. According to the *factor property,* s has an $\xrightarrow{\Diamond}$–prolongation to $[t]$ as well. Thus $s \in U_t$. This shows $U_u \subseteq U_t$. Since $\prec$ is true, there exists $t' \in [t]$ having no $\xrightarrow{\Diamond}$–successor contained in $[u]$. Let $s' \in [s_0]$ be a $\xrightarrow{\Diamond}$–predecessor of t'. Then $s' \in U_t$, but $s' \notin U_u$; for otherwise s' would factor through $[t]$ because of Proposition 2.5, and the corresponding element would have to equal t' because of Proposition 2.2. Contradiction! Therefore, $U_u \subset U_t$.

   Now suppose that $[t] \not\prec [u]$. Then, because $\prec$ is faithfully induced by $\xrightarrow{\Diamond}$, there exists a point $s' \in [s_0]$ having an $\xrightarrow{\Diamond}$–successor contained in $[u]$, but no $\xrightarrow{\Diamond}$ –successor contained in $[t]$. That is, $U_u \not\subseteq U_t$, hence $U_u \not\subset U_t$ in particular. This shows the opposite direction.

Next we obtain a suitable *Truth Lemma,* where we use the same notations as above.

Lemma 1 (Truth Lemma). *For all $\beta \in$ WFF, $s \in [s_0]$, and $t \in I$ such that t_s exists, we have*

$$s, U_t \models_{\widetilde{\mathcal{M}}} \beta \iff \beta \in t_s .$$

Proof. We only consider the basic case 'β a nominal' of the structural induction in detail, and give some comments on the other cases afterwards.

Let $\beta = \mathfrak{n} \in$ NNS. Then,

$$\begin{aligned}
s, U_t \models_{\widetilde{\mathcal{M}}} \beta &\iff \widetilde{V}(\mathfrak{n}) = s, U_t && \text{(Def. 2)}\\
&\iff s = s_{\mathfrak{n}} \text{ and } U_t = U_{V(\mathfrak{n})} && \text{(Def. of } \widetilde{V}\text{)}\\
&\iff s = s_{\mathfrak{n}} \text{ and } [t] = [V(\mathfrak{n})] && \text{(Prop. 3)}\\
&\iff s = s_{\mathfrak{n}} \text{ and } \mathfrak{n} \in V(\mathfrak{n}) = V(\mathfrak{n})_{s_{\mathfrak{n}}} = t_{s_{\mathfrak{n}}}\\
&\iff \beta = \mathfrak{n} \in t_s .
\end{aligned}$$

Note that we used once again that $\prec$ is injectively induced by $\xrightarrow{\Diamond}$, for the bottom–to–top direction of the last equivalence. This property is also applied in case $\beta = K\gamma$, where the *cross property* comes into play as well.

In case of a propositional variable Axiom 13 plays its part. The other boolean cases and the case of an $@_{\mathfrak{n}}$–formula, too, follow straightforwardly from the induction hypothesis. Finally, in case $\beta = \Box\gamma$ one has to take into account Proposition 3 for a second time.

Since $\neg\gamma$ is contained in t_s for some $s \in [s_0]$ and $t \in I$, Lemma 1 immediately gives us the following *Completeness Theorem.*

Theorem 1 (Completeness). *Let* **S** *be the logical system determined by the above axiom schemata and rules. Then every non–***S***–derivable formula γ fails to hold at some neighbourhood situation of some hybrid subset space.*

5 Dense Flows of Time

Most of our work towards the desired completeness theorem for dense subset spaces has been done already. According to [2], Theorem 7.29 (see also the end of Section 3), we only have left to add a hybrid formula expressing density. And such a formula really exists:

23. $\Diamond\mathfrak{n} \rightarrow \Diamond\Diamond\mathfrak{n}$

Note that Axiom 23 is the 'converse' of Axiom 14; moreover, this schema represents the hybrid version of the modal density correspondent, $\Diamond\alpha \rightarrow \Diamond\Diamond\alpha$.

Let **D** be **S** plus Axiom 23. Then:

Theorem 2 (Soundness and Completeness for D). *A formula α is* **D***–derivable, iff it is valid in every hybrid subset space based on a dense subset frame.*

It turns out that the poperty of subset frames to be linear (cf Definition 3) can be captured by a hybrid formula as well, viz that one expressing *trichotomy* of the composite relation $\xrightarrow{L} \circ \xrightarrow{\Diamond}$:

24. $@_{\mathfrak{m}} L \Diamond \mathfrak{n} \vee @_{\mathfrak{m}} \mathfrak{n} \vee @_{\mathfrak{n}} L \Diamond \mathfrak{m}$

This follows easily from our construction in Section 4. So, if we let **L** be **D** plus Axiom 24, then we obtain a soundness and completeness result with respect to the class of dense *and* linear subset frames.

Theorem 3 (Soundness and Completeness for L). *A formula α is* **L**–*derivable, iff it is valid in every hybrid subset space based on a subset frame that is dense and linear.*

This game can be played further, taking into account other interesting properties of time. We still touch on that one of *branching* here, representing the counterpart to the case of linear time dealt with in Theorem 3. The corresponding substitute of Axiom 24 reads

$$\Diamond \mathfrak{n} \wedge \Diamond \mathfrak{m} \rightarrow \Diamond \left((\mathfrak{n} \wedge \Diamond \mathfrak{m}) \vee (\mathfrak{m} \wedge \Diamond \mathfrak{n}) \vee (\mathfrak{n} \wedge \mathfrak{m}) \right),$$

expressing *no branching to the right for states.* It is not hard to see that, in fact, *treelike spaces,* cf [6], are characterized in this way.

We finish now our treatment of hybrid completeness for dense subset spaces and turn to a concluding discussion of our approach.

6 Concluding Remarks

We applied above hybrid logic, a recently developed extension of modal logic by names of states and corresponding jump operators, to solve the problem of completely axiomatizing some important classes of structures modelling knowledge and time. In particular, we proved a completeness theorem for subset spaces based on dense frames.

It may be supposed that the systems considered in this paper are *decidable.* However, proving this claim requires methods quite different from those applied usually in modal logics of knowledge. Note that each of the above systems obviously lacks the finite model property, in particular. Maybe certain decidable fragments of first–order temporal logic can be helpful for our purposes; cf [12]. In this case we would get a decision procedure for the *MLSS*–theory of dense subset frames, too.

In case of *reflexive* flows of time the question of decidability is much easier to deal with. Actually, the hybrid satisfiability problem with respect to subset spaces based on linear frames turns out to be NP–complete then; cf [9].

Hybrid logic offers a lot of interesting features the significance of which to spaces of knowledge states should be clarified by future research, for example, *state variables* and corresponding binders (eg, $\downarrow$; cf [3]). Because of the success of the first steps undertaken in [9] and the present paper, respectively, we believe that hybrid logic will also contribute a lot to those reasoning tasks where topological concepts are involved in.

Acknowledgement

I thank the referees very much for their helpful comments on the first version of this paper.

References

1. Patrick Blackburn. Representation, Reasoning, and Relational Structures: a Hybrid Logic Manifesto. *Logic Journal of the IGPL*, 8:339–365, 2000.
2. Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 2001.
3. Patrick Blackburn and Miroslava Tzakova. Hybrid Languages and Temporal Logic. *Logic Journal of the IGPL*, 7(1):27–54, 1999.
4. Andrew Dabrowski, Lawrence S. Moss, and Rohit Parikh. Topological Reasoning and The Logic of Knowledge. *Annals of Pure and Applied Logic*, 78:73–110, 1996.
5. Ronald Fagin, Joseph Y. Halpern, Yoram Moses, and Moshe Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995.
6. Konstantinos Georgatos. Knowledge on Treelike Spaces. *Studia Logica*, 59:271–301, 1997.
7. Bernhard Heinemann. On Sets Growing Continuously. In C. Pandu Rangan, V. Raman, and R. Ramanujam, editors, *Foundations of Software Technology & Theoretical Computer Science, 19th Conference, FST&TCS 1999*, volume 1738 of *Lecture Notes in Computer Science*, pages 420–431, Berlin, 1999. Springer.
8. Bernhard Heinemann. Modelling Change with the Aid of Knowledge and Time. In R. Freivalds, editor, *Fundamentals of Computation Theory, 13th International Symposium, FCT 2001*, volume 2138 of *Lecture Notes in Computer Science*, pages 150–161, Berlin, 2001. Springer.
9. Bernhard Heinemann. A Hybrid Treatment of Evolutionary Sets. In C. A. Coello Coello, A. de Albornoz, L. E. Sucar, and O. Cairó Battistutti, editors, *MICAI'2002: Advances in Artificial Intelligence*, volume 2313 of *Lecture Notes in Artificial Intelligence*, pages 204–213, Berlin, 2002. Springer.
10. Bernhard Heinemann. Linear Tense Logics of Increasing Sets. *Journal of Logic and Computation*, 12(4):583–606, 2002.
11. Jaakko Hintikka. *Knowledge and Belief*. Cornell University Press, Ithaca, NY, 1962.
12. Ian Hodkinson, Frank Wolter, and Michael Zakharyaschev. Decidable Fragments of First–Order Temporal Logics. *Annals of Pure and Applied Logic*, 106(1–3):85–134, 2000.
13. J.-J. Ch. Meyer and W. van der Hoek. *Epsitemic Logic for AI and Computer Science*, volume 41 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, Cambridge, 1995.
14. Lawrence S. Moss and Rohit Parikh. Topological Reasoning and The Logic of Knowledge. In Y. Moses, editor, *Proceedings of the 4th Conference on Theoretical Aspects of Reasoning about Knowledge (TARK 1992)*, pages 95–105, San Francisco, CA, 1992. Morgan Kaufmann.
15. M. Angela Weiss and Rohit Parikh. Completeness of Certain Bimodal Logics for Subset Spaces. *Studia Logica*, 71:1–30, 2002.

The Complexity of the Inertia[⋆]

Thanh Minh Hoang[1] and Thomas Thierauf[2,⋆⋆]

[1] Abt. Theoretische Informatik, Universität Ulm, D-89069 Ulm
[2] FB Elektronik und Informatik, FH Aalen, D-73430 Aalen
{hoang,thierauf}@informatik.uni-ulm.de

Abstract. The *inertia* of a square matrix A is defined as the triple $(i_+(A), i_-(A), i_0(A))$, where $i_+(A)$, $i_-(A)$, and $i_0(A)$ are the number of eigenvalues of A, counting multiplicities, with positive, negative, and zero real part, respectively. A hard problem in Linear Algebra is to compute the inertia. No method is known to get the inertia of a matrix exactly in general. In this paper we show that the inertia is hard for **PL** (*probabilistic logspace*) and in some cases the inertia can be computed in **PL**. We extend our result to some problems related to the inertia. Namely, we show that matrix stability is complete for **PL** and the inertia of symmetric matrices can be computed in **PL**.

1 Introduction

A fundamental topic in linear algebra is the study of equivalence relations between matrices that naturally arise in theory and in applications. In computer science, we are interested in finding efficient algorithms to decide equivalence, or to construct canonical forms of a matrix for the relation under consideration. More general, we are interested in the *computational complexity* of these and related problems.

Most of these problems can be solved within certain logspace counting classes, all of which are contained in the parallel complexity class (uniform) $\mathbf{NC}^2$. In fact, the logspace counting class **GapL** [AO96] seems to capture the complexity of a lot of algebraic problems quite naturally. **GapL** is the extension of #**L** to integers in the same way as #**P** [Val79b,Val79a] can be extended to **GapP** [FFK94] in the polynomial time setting. The break-through result for **GapL** was that it precisely captures the complexity of the determinant of an integer matrix [Ber84,Dam91,Tod91,Vin91,Val92].

The verification of the value of a **GapL**-function defines the complexity class $\mathbf{C_=L}$. For example, the singularity problem (deciding whether a matrix is singular) is complete for $\mathbf{C_=L}$, because this is asking whether the determinant of a matrix is zero.

Inequalities on **GapL**-functions define the complexity class **PL**. For example, the problem to decide whether the determinant of a matrix is positive, is complete for **PL**.

[⋆] Supported by the Deutsche Forschungsgemeinschaft
[⋆⋆] Part of the work done at Universität Ulm.

The computational problems over matrices like testing similarity, equivalence, and congruence are located in logspace counting classes like $\mathbf{AC}^0(\mathbf{C}_=\mathbf{L})$ (the $\mathbf{AC}^0$-closure of $\mathbf{C}_=\mathbf{L}$) or **PL**. We describe these results in more detail.

Similarity. Two matrices A and B are *similar*, if there is a nonsingular matrix S such that $A = S^{-1}BS$. Santha and Tan [ST98] observed that testing similarity is in $\mathbf{AC}^0(\mathbf{C}_=\mathbf{L})$. Testing similarity is actually complete for this class [HT00]. A and B are similar iff they have the same invariant factors. The invariant factors can be computed in $\mathbf{AC}^0(\mathbf{GapL})$ and are hard for **GapL** [HT01].

Equivalence. Two matrices A and B are *equivalent*, if there exist nonsingular matrices P and Q, such that $A = PBQ$. A simple characterization of equivalence is that A and B have the same rank. Testing the equivalence of two matrices is complete for $\mathbf{AC}^0(\mathbf{C}_=\mathbf{L})$ [ABO99,HT00], as well as verifying one bit of the rank of matrix [ABO99].

Congruence. Two symmetric real matrices A and B are *congruent (via a real matrix)*, if there exists a nonsingular real matrix S such that $A = SBS^T$. By Congruence we denote the problem of testing the congruence. Congruence is hard for $\mathbf{AC}^0(\mathbf{C}_=\mathbf{L})$ [HT00]. An upper bound for Congruence is an open problem in [HT00]. In this paper we show that Congruence $\in$ **PL** by considering the inertia of symmetric matrices.

The computational complexity of the inertia and its related problems is the main topic of this paper. In Section 3.2, we use the *Routh-Hurwitz Theorem* to show that the inertia of a matrix (under the restrictions of the Routh-Hurwitz Theorem) can be computed in **PL**. In Section 3.3 we show that the inertia is hard for **PL**.

An alternative way to compute the inertia of a matrix could be to determine all the roots of the characteristic polynomial of the given matrix. With the $\mathbf{NC}^2$-algorithm of Neff and Reif [Nef94,NR96] these roots can be approximated to some precision [ABO]. However, it is not clear to what precision we have to approximate a root in order to tell it apart from zero. This result is different from our approach.

We also consider the verification of the inertia. That is, for matrix A and integers $\mathfrak{p}$, $\mathfrak{n}$, and $\mathfrak{z}$, one has to decide whether $(\mathfrak{p}, \mathfrak{n}, \mathfrak{z})$ is the inertia of A. We show in Section 3 that for certain matrices the verification is complete for **PL**.

A system of differential equations is stable iff its coefficient matrix is stable (matrix whose eigenvalues have negative real parts). Therefore, the study of stable matrices has a long-standing history and it is an important topic in Linear Algebra. We prove in Section 4 that the problem of deciding whether all eigenvalues of a matrix have positive real parts is complete for **PL**. A matrix has no eigenvalues with negative real part is called *positive semistable*. We show in Section 4 that the problem to decide whether a matrix is positive semistable is in $\mathbf{AC}^0(\mathbf{GapL})$ and is hard for **PL**.

Finally, in Section 5 we prove that the inertia of a *symmetric* integer matrix can be computed in **PL**. It follows that the congruence of two matrices can be decided in **PL**. Note that there are deterministic algorithms for the inertia of symmetric integer matrices see for example [For00].

2 Preliminaries

We assume familiarity with some basic notions of complexity theory and linear algebra. We refer the readers to the papers [ABO99,AO96] for more details and properties of the considered complexity classes, and to the textbooks [Gan77,HJ91,HJ85] for more background in linear algebra.

Complexity Classes. For a nondeterministic Turing machine M, let gap_M denote the difference between the number of accepting and rejecting computation paths of M on input x. The function class **GapL** is defined as the class of all functions $gap_M(x)$ such that M is a nondeterministic logspace bounded Turing machine M.

It is easy to see that **GapL** is closed under addition, subtraction, and multiplication. Allender, Arvind, and Mahajan [AAM99] showed that **GapL** is closed under composition. Even stronger, they showed that the determinant of a matrix A where each entry of A is computed in **GapL** can be computed in **GapL**.

A set S is in $\mathbf{C_{=}L}$, if there exists a function $f \in \mathbf{GapL}$ such that for all x we have $x \in S \iff f(x) = 0$. A set S is in **PL** if there is a function $f \in \mathbf{GapL}$ such that for all x we have $x \in S \iff f(x) > 0$. Ogihara [Ogi98] showed that **PL** is closed under logspace Turing reductions.

By $\mathbf{AC}^0(\mathbf{C_{=}L})$, $\mathbf{AC}^0(\mathbf{PL})$, and $\mathbf{AC}^0(\mathbf{GapL})$ we denote the class of sets that are $\mathbf{AC}^0$-reducible to a set in $\mathbf{C_{=}L}$, **PL**, respectively a function in **GapL**. All these classes are contained in $\mathbf{TC}^1$, a subclass of $\mathbf{NC}^2$. The known relationships among these classes are as follows:

$$\mathbf{C_{=}L} \subseteq \mathbf{AC}^0(\mathbf{C_{=}L}) \subseteq \mathbf{AC}^0(\mathbf{PL}) = \mathbf{PL} \subseteq \mathbf{AC}^0(\mathbf{GapL}) \subseteq \mathbf{TC}^1 \subseteq \mathbf{NC}^2.$$

Unless otherwise stated, all reductions in this paper are logspace many-one.

Linear Algebra. Let M_n be the set of $n \times n$ integer matrices. For $A \in M_n$ we denote the *characteristic polynomial* of A by $\chi_A(x)$, that is $\chi_A(x) = \det(xI - A)$ is a degree n polynomial, $\deg(\chi_A) = n$. The *companion matrix* of the polynomial $p(x) = x^n + \alpha_1 x^{n-1} + \cdots + \alpha_n$ is the matrix $P \in M_n$, where the last column is $(-\alpha_n, \ldots, -\alpha_1)^T$, all entries on the lower subdiagonal are 1. All the other elements are zero. The property of P we use is that $\chi_P(x) = p(x)$.

The *inertia* of a matrix $A \in M_n$ is defined as the triple $(i_+(A), i_-(A), i_0(A))$, where $i_+(A)$, $i_-(A)$, and $i_0(A)$ are the number of eigenvalues of A, counting multiplicities, with positive, negative, and zero real part, respectively. Note that $i_+(A), i_-(A), i_0(A)$ are nonnegative integers and the sum of these is exactly n.

Matrix A is called *positive stable*, if $i(A) = (n, 0, 0)$, and *negative stable*, if $i(A) = (0, n, 0)$. Furthermore, A is called as *positive semistable* if $i_-(A) = 0$. In case that A is *real symmetric* all eigenvalues of A are real and the word *"stable"* will be replaced by *"definite"*.

For square matrices $A = (a_{i,j}) \in M_n$ and $B \in M_m$, the *Kronecker product* $A \otimes B$ is defined as the matrix $(a_{i,j}B) \in M_{nm}$. The *Kronecker sum* $A \oplus B$ is defined as the matrix $A \otimes I_m + I_n \otimes B \in M_{nm}$, where $I_n \in M_n$ and $I_m \in M_m$ are identity matrices. If $\lambda_1, \ldots, \lambda_n$ and $\mu_1, \ldots, \mu_m$ are the eigenvalues of A and B, respectively, then the eigenvalues of $A \otimes B$ are $\lambda_k \mu_l$, and the eigenvalues of $A \oplus B$ are $\lambda_k + \mu_l$, for all $1 \le k \le n$ and $1 \le l \le m$.

Problems. We define some natural problems in linear algebra that we are considering. Unless otherwise specified, our domain for the algebraic problems are the integers. The two following functions are complete for **GapL** [ABO99,ST98].

PowerElement: given $A \in M_n$ and m, compute $(A^m)_{1,n}$.

Determinant: given $A \in M_n$, compute $\det(A)$.

For each of them, we define the corresponding *verification problem* as the graph of the corresponding function: for a fixed function $f(x)$, define v-f as the set of all pairs (x, y) such that $f(x) = y$. This yields v-PowerElement and v-Determinant. They are known to be complete for $\mathbf{C_{=}L}$.

We denote by PosPowerElement the problem of deciding whether one element of the power of a matrix is positive, and by PosDeterminant the problem of deciding whether the determinant of a matrix is positive. These problems are complete for **PL**.

We define Inertia to be the problem of computing one bit of $i(A)$ (with respect to some fixed coding). That is,

$$\textsc{Inertia} = \{(A, k, b) |\ \text{the } k\text{-th bit of } i(A) \text{ is } b\}.$$

By v-Inertia we denote the problem of verifying the value of $i(A)$.

PosStable and PosSemistable are the sets of all positive stable, semistable matrices, respectively. PosDefinite and PosSemidefinite are the sets of all positive definite, semidefinite matrices, respectively.

3 The Inertia

3.1 The Routh-Hurwitz Theorem

The Routh-Hurwitz Theorem (see [Gan77], Volume II, Chapter XV) provides a method for determining the number of roots in the right half-plane of a given real polynomial. Since the roots of the characteristic polynomial $\chi_A(x)$ are the eigenvalues of the matrix A, we can compute the inertia of A by applying the Routh-Hurwitz method to $\chi_A(x)$.

Let $A \in M_n$. Consider the characteristic polynomial of A

$$\chi_A(x) = x^n + c_1 x^{n-1} + c_2 x^{n-2} + \cdots + c_n.$$

Define $c_0 = 1$. The *Routh-Hurwitz matrix* $\Omega(A) = (\omega_{i,j}) \in M_n$ is defined as

$$\Omega(A) = \begin{pmatrix} c_1 & c_3 & c_5 & c_7 & \cdots & 0 \\ c_0 & c_2 & c_4 & c_6 & \cdots & 0 \\ 0 & c_1 & c_3 & c_5 & \cdots & 0 \\ 0 & c_0 & c_2 & c_4 & \cdots & 0 \\ \vdots & & & & \ddots & \vdots \\ 0 & 0 & 0 & 0 & \cdots & c_n \end{pmatrix}.$$

That is, the diagonal elements of $\Omega(A)$ are $\omega_{i,i} = c_i$. In the i-th column, the elements above the diagonal are $\omega_{i-1,i} = c_{i+1}$, $\omega_{i-2,i} = c_{i+2}$, $\ldots$ until we reach either the first row $\omega_{1,i}$ or c_n. In the latter case, the remaining entries are filled with zeros. The elements below $\omega_{i,i}$ are $\omega_{i+1,i} = c_{i-1}$, $\omega_{i+2,i} = c_{i-2}$, ..., c_1, c_0, 0, 0, ... down to the last row $\omega_{n,i}$.

The successive leading principal minors D_i of $\Omega(A)$ are called the *Routh-Hurwitz determinants*, they are

$$D_1 = c_1, \quad D_2 = \det\begin{pmatrix} c_1 & c_3 \\ c_0 & c_2 \end{pmatrix}, \ \cdots, \quad D_n = \det(\Omega(A)).$$

Theorem 3.1 (Routh-Hurwitz). *If $D_n \neq 0$, then the number of roots of the polynomial $\chi_A(x)$ in the right half-plane is determined by the formula*

$$i_+(A) = V(1, D_1, \frac{D_2}{D_1}, \ldots, \frac{D_n}{D_{n-1}}),$$

where $V(x_1, x_2, \ldots)$ computes the number of sign alternations in the sequence of numbers $x_1, x_2, \ldots$. For the calculation of the values of V, for every group of p successive zero Routh-Hurwitz determinants (p is always odd!)

$$D_s \neq 0, \ D_{s+1} = \cdots = D_{s+p} = 0, \ D_{s+p+1} \neq 0$$

we have to set $V(\frac{D_s}{D_{s-1}}, \frac{D_{s+1}}{D_s}, \ldots, \frac{D_{s+p+2}}{D_{s+p+1}}) = h + \frac{1-(-1)^h\varepsilon}{2}$ *, where* $p = 2h-1$ *and* $\varepsilon = \operatorname{sign}(\frac{D_s}{D_{s-1}}\frac{D_{s+p+2}}{D_{s+p+1}})$*. For* $s = 1$*,* $\frac{D_s}{D_{s-1}}$ *is to be replaced by* D_1*; and for* $s = 0$*, by* c_0*.*

Let us discuss the case when $D_n = 0$. It is known that $D_n = 0$ iff $\chi_A(x)$ has a pair of opposite roots x_0 and $-x_0$ (see [Gan77]). Define

$$p_1(x) = x^n + c_2x^{n-2} + c_4x^{n-4} + \cdots \quad \text{and} \quad p_2(x) = c_1x^{n-1} + c_3x^{n-3} + \cdots.$$

Then $\chi_A(x) = p_1(x) + p_2(x)$ and $p_1(x_0) = p_2(x_0) = 0$. Therefore, x_0 is also a root of the greatest common divisor $g(x)$ of $p_1(x)$ and $p_2(x)$. We can write $\chi_A(x) = g(x)\chi_A^*(x)$, where the polynomial $\chi_A^*(x)$ has no pair of opposite roots, i.e. the Routh-Hurwitz matrix of $\chi_A^*(x)$ is nonsingular. Let B be the companion matrix of $g(x)$ and C be the companion matrix of $\chi_A^*(x)$. Then we have

$$i(A) = i(B) + i(C).$$

Note that all nonzero-eigenvalues of B are pairs of opposite values. The Routh-Hurwitz method does not work in the case where B has some opposite eigenvalues *on* the imaginary axis, and no method is known to get the exact number of roots of a polynomial on an axis (to the best of our knowledge).

However, there are methods to determine the number of *distinct* roots of a polynomial on an axis, and we will show below how to use these methods to solve at least some cases where $D_n = 0$.

Let P be the companion matrix of a polynomial $p(x)$, where $\deg(p(x)) = n$. The *Hankel matrix* $H = (h_{i,j}) \in M_n$ associated with $p(x)$ is defined as $h_{i,j} =$

trace(P^{i+j-2}), for $i, j = 1, \ldots, n$, where trace(P^{i+j-2}) is the sum of all diagonal elements of P^{i+j-2}. Note that H is symmetric. By sig(H) we denote the *signature* of H, that is the difference between $i_+(H)$ and $i_-(H)$. The following Theorem can be found in Volume II, Chapter XV of [Gan77].

Theorem 3.2. *1) The number of distinct* real *roots of $p(x)$ is equal* sig(H)*.*
2) The number of all *distinct roots of $p(x)$ is equal to the rank of H.*

3.2 Upper Bounds

We consider the complexity to compute the inertia via Theorem 3.1. The first step is to compute all the coefficients c_i of $\chi_A(x)$ and from these all Routh-Hurwitz determinants D_i, for $i = 1, \ldots, n$. Since the coefficients $c_1, \ldots, c_n$ are computable in **GapL**, each of the determinants $D_1, \ldots, D_n$ can be computed in **GapL** as well [AAM99].

If $D_n \neq 0$, i.e. $\Omega(A)$ is nonsingular, we can compute $i_+(A)$ by using the formulas from Theorem 3.1: a logspace machine with a **PL** oracle can ask, for each of the determinants $D_1, \ldots, D_n$, if it is positive, negative, or zero. Because $i_-(A) = i_+(-A)$, we can apply the same method to compute $i_-(A)$ and get $i_0(A) = n - i_+(A) - i_-(A)$. Hence $i(A)$ can be computed in **PL**.

Theorem 3.3. *The inertia of a matrix A with the property that $\Omega(A)$ is nonsingular can be computed in* **PL***.*

Let us consider the case when $D_n = 0$, i.e. when $\Omega(A)$ is singular. We decompose $\chi_A(x) = g(x)\chi_A^*(x)$, as described in the previous section. Recall that $g(x)$ is the greatest common divisor of two polynomials $p_1(x)$ and $p_2(x)$. Therefore the coefficients of $g(x)$ can be computed as the solution of a system of linear equations (see [Koz91]), which can be done in $\mathbf{AC}^0(\mathbf{GapL})$ (see [ABO99]). It follows that we can compute the polynomial $\chi_A^*(x)$ in $\mathbf{AC}^0(\mathbf{GapL})$ as well. In other words, each of the elements of the companion matrices B (of $g(x)$) and C (of $\chi_A^*(x)$) can be computed in $\mathbf{AC}^0(\mathbf{GapL})$.

There is no method to compute $i(B)$ in general. However, in some cases, when $g(x)$ is easy, we can do so anyway. Suppose for example that

$$g(x) = x^t, \text{ for some } t \geq 0.$$

Equivalently we can say that B (and hence A) has *no opposite nonzero-eigenvalues*. Then it is clear that $i(B) = (0, 0, t)$, and hence $i(A) = (0, 0, t) + i(C)$.

Note that the decision whether A has no opposite nonzero-eigenvalues is in $\mathbf{coC_=L}$: with the greatest t such that x^t is a divisor of $\chi_A(x)$ (it is possible that $t = 0$) we can decide whether the Routh-Hurwitz matrix associated with the polynomial $\frac{\chi_A(x)}{x^t}$ is nonsingular.

Corollary 3.4. *The inertia of a matrix with no opposite nonzero-eigenvalues can be computed in* **PL***.*

We can considerably extend Corollary 3.4 to the following theorem.

Theorem 3.5. *The inertia of a matrix A with the property that*
1) A has all *opposite eigenvalues on the imaginary axis,* or
2) A has no *opposite eigenvalues on the imaginary axis,*
can be computed in $\mathbf{AC}^0(\mathbf{GapL})$.

Proof. Assume that the condition on A is fulfilled. Let B be again the companion matrix of $g(x)$. The triple $i(B)$ can be easily computed. In the case 1) we have $i(B) = (0, 0, \deg(g(x))$ and in the case 2) we have $i(B) = (\frac{1}{2}\deg(g(x)), 0, \frac{1}{2}\deg(g(x)))$. Thus we can compute $i(A)$ by adding $i(B)$ to $i(C)$.

We show how to check the condition on A by using Theorem 3.2.

Since Theorem 3.2 deals with the real axis instead of the imaginary axis, we first turn $g(x)$ by 90°: consider the matrix $E = \left(\begin{smallmatrix} 0 & -1 \\ 1 & 0 \end{smallmatrix}\right)$. Its eigenvalues are $+i$ and $-i$. Define $D = B \otimes E$. The eigenvalues of D are $i\lambda_k(B)$ and $-i\lambda_k(B)$ where $\lambda_k(B)$ runs through all eigenvalues of B. It follows that the number of distinct purely imaginary eigenvalues of B is the same as the number of distinct real eigenvalues of D.

Finally, let H be the Hankel matrix of $\chi_D(x)$. From Theorem 3.2 we have

$$\begin{aligned} i_+(B) = 0 &\iff \mathrm{rank}(H) = \mathrm{sig}(H), \\ i_0(B) = 0 &\iff \mathrm{sig}(H) = 0. \end{aligned}$$

The conditions on the right-hand side can be decided in $\mathbf{AC}^0(\mathbf{GapL})$. This proves the theorem. □

Because of the closure properties of **PL** and $\mathbf{AC}^0(\mathbf{GapL})$, we get the same upper bounds for the verification of the inertia.

3.3 Lower Bounds

Theorem 3.6. Inertia *and* v-Inertia *are hard for* **PL**.

Proof. We reduce PosPowerElement, a complete problem for **PL**, to Inertia and v-Inertia. Let $A \in M_n$ be an input for PosPowerElement. One has to decide whether $(A^m)_{1,n} > 0$ for a given $m > 1$.

There is a reduction from matrix powering to the characteristic polynomial which is shown in [HT00] (see also [HT01] and [HT02]): given A, m and a, one can construct a matrix B in $\mathbf{AC}^0$ such that

$$(A^m)_{1,n} = a \iff \chi_B(x) = x^{N-2m-1}\left(x^{2m+1} - a\right),$$

where $N = m(n+d) + n$, and d is the number of nonzero-elements in A.

The eigenvalues of B are the roots of $\chi_B(x)$. We consider the case when $a = (A^m)_{1,n} \neq 0$. The roots of $x^{2m+1} - a$ are the corners of a regular $(2m+1)$-gon inscribed in a circle of radius $a^{\frac{1}{2m+1}}$ with its center at the origin. Since $2m+1$ is odd, none of these roots lies on the imaginary axis. This implies that $i_0(B) = N - (2m+1)$, and one of $i_+(B)$ and $i_-(B)$ is m and the other is $m+1$. Moreover, these values depend on the sign of a. Namely, if $a > 0$, we have

$$i_+(B) = \begin{cases} m+1 & \text{if } 2m+1 \equiv 1 \pmod 4, \\ m & \text{if } 2m+1 \equiv 3 \pmod 4. \end{cases} \tag{1}$$

Note in particular that $i_+(B)$ in (1) is always odd. Analogously, $i_+(B)$ is even if $a < 0$. In the case where $(A^m)_{1,n} = 0$, we have $i(B) = (0, 0, N)$.

In summary, we can compute values $\mathfrak{p}$, $\mathfrak{n}$, $\mathfrak{z}$ in logspace such that

$$\begin{aligned}(A^m)_{1,n} > 0 &\Longleftrightarrow i(B) = (\mathfrak{p}, \mathfrak{n}, \mathfrak{z}) \\ &\Longleftrightarrow i_+(B) = \text{odd}\end{aligned}$$

This proves the theorem. □

Note also that B in the above proof has no pair of opposite nonzero-eigenvalues. Therefore B fulfills the condition of Corollary 3.4.

Corollary 3.7. *The computation and the verification of the inertia of a matrix with no opposite nonzero-eigenvalues are complete for* **PL**.

4 Stability

Theorem 4.1. PosStable $\in$ **PL** *and* PosSemistable $\in$ $\mathbf{AC}^0(\mathbf{GapL})$

Proof. A is positive stable iff all the Routh-Hurwitz determinants of the matrix $\Omega(-A)$ are positive. Hence, positive stability of A can be decided in **PL**.

If $\Omega(A)$ is nonsingular, then PosSemistable $\in$ **PL** by Theorem 3.3. So assume that $\Omega(A)$ is singular. As described in Section 3.1, we decompose $\chi_A(x) = g(x)\chi^*_A(x)$ in $\mathbf{AC}^0(\mathbf{GapL})$. Let B and C be the companion matrices of $g(x)$ and $\chi^*_A(x)$, respectively. Then A is positive semistable iff B is positive semistable and C is positive stable. Matrix B is positive semistable iff all eigenvalues of B are on the imaginary axis. Now the result follows from Theorem 3.5. □

Now we consider the hardness of the stability problems. A matrix A is nonsingular iff AA^T is positive definite. AA^T can be computed in $\mathbf{NC}^1$. Therefore, PosDefinite is hard for $\mathbf{coC_=L}$ under $\mathbf{NC}^1$ many-one reductions.

Corollary 4.2. PosDefinite *is hard for* $\mathbf{coC_=L}$.

Theorem 4.3. PosStable *and* PosSemistable *are hard for* **PL**.

Proof. By NegDeterminant we denote the set of all matrices with negative determinant. Note that NegDeterminant is complete for **PL**. We construct a reduction from NegDeterminant to PosStable as follows.

Let $A \in M_n$. Let $d_{1,1}, \ldots, d_{n,n}$ be the diagonal elements of A^TA. The *Hadamard Inequality* states that $|\det(A)| \leq (d_{1,1} \cdots d_{n,n})^{1/2}$. W.l.o.g. we can assume that the input matrix A is a 0-1 matrix and that no row or column of A has more than two 1's in it [All02]. Therefore $d_{i,i} \leq 2$ for all i. By the Hadamard Inequality we get the following bound for $\det(A)$:

$$-2^n < \det(A) < 2^n.$$

Define $t = \lceil \frac{n}{2m+1} \rceil (2m + 1)$, for an integer $m \geq 1$. Since $n \leq t$, we have $\det(A) + 2^t > 0$ and

$$\det(A) < 0 \Longleftrightarrow \det(A) + 2^t < 2^t. \tag{2}$$

Lemma 4.4. *We can construct a matrix $B \in M_k$ and m such that*

$$(B^m)_{1,k} = \det(A) + 2^t. \tag{3}$$

Note that m depends on t, and we defined t in terms of m. This makes the construction a bit tricky. We prove the lemma below.

Define $b = (B^m)_{1,k}$. We further reduce B to a matrix C such that

$$\chi_C(x) = x^{N-2m-1}\left(x^{2m+1} - b\right),$$

where $N = m(k+d) + k$, and d is the number of elements different from zero of B [HT00] (see also [HT01] and [HT02]). This is an $\mathbf{AC}^0$-reduction.

As explained in Theorem 3.6, matrix C has $N - 2m - 1$ eigenvalues zero and $2m+1$ eigenvalues as the roots of $x^{2m+1} - b$. The latter $2m+1$ eigenvalues of C are complex and lie on the circle of radius $r = b^{\frac{1}{2m+1}}$ (with the origin as center). Since $b > 0$, the eigenvalue with the largest real part is $\lambda_{\max}(C) = r$.

We shift the eigenvalues of C by $s = 2^{\frac{t}{2m+1}} = 2^{\lceil \frac{n}{2m+1} \rceil}$. That is, define the matrix $D = -C + sI$. The eigenvalue of C with the largest real part becomes the eigenvalue of D with the smallest real part: $\lambda_{\min}(D) = -r + s$. So we get

$$b < 2^t \iff r < s \iff \lambda_{\min}(D) > 0. \tag{4}$$

By (2) and (4) we have $A \in$ NegDeterminant $\iff D \in$ PosStable.

An analogous argument reduces the set of matrices with nonpositive determinants (a **PL**-complete set) to PosSemistable □

Proof of Lemma 4.4. Since PowerElement is complete for **GapL**, we can compute $B_0 \in M_l$ and an exponent m in logspace such that $(B_0^m)_{1,l} = \det(A)$.

Define a $(m+1) \times (m+1)$ block matrix F with m times B_0 on the first upper subdiagonal, all other blocks are zero.

Define $S = \left(\begin{smallmatrix} s^2 & s^3 \\ 0 & 0 \end{smallmatrix}\right)$, where $s = 2^{\lceil \frac{n}{2m+1} \rceil}$. It is easily to get $S^m = \left(\begin{smallmatrix} s^{2m} & s^{2m+1} \\ 0 & 0 \end{smallmatrix}\right)$ and $s^{2m+1} = 2^t$.

We define an $l(m+1) \times 2$ matrix T whose elements at the position $(1,1)$ and $(l(m+1), 2)$ are 1 and all the others are zero.

Finally, for $k = l(m+1) + 2$ we define

$$B = \begin{pmatrix} F & FT + TS \\ \mathbf{0} & S \end{pmatrix},$$

and claim that the matrix B fulfills (3). From the powers of B we get

$$B^m = \begin{pmatrix} F^m & F^mT + 2F^{m-1}TS + 2F^{m-2}TS^2 + \cdots + 2FTS^{m-1} + TS^m \\ \mathbf{0} & S^m \end{pmatrix}.$$

In particular, for each $1 \le i \le m$, F^i has a very simple form: on its i-th upper subdiagonal are purely B^i and all the other block-elements are zeromatrix. Furthermore, it is not hard to see that $F^{m-i}TS^i = \mathbf{0}$ for all $i < m$. Thus

$$B^m = \begin{pmatrix} F^m & F^mT + TS^m \\ \mathbf{0} & S^m \end{pmatrix}.$$

Now, it is not hard to see that $(B^m)_{1,k} = \det(A) + 2^t$. This proves the lemma.

5 The Congruence of Symmetric Matrices

Recall that all the eigenvalues of a symmetric (real) matrix A are real. Therefore, if we decompose $\chi_A(x) = g(x)\chi_A^*(x)$ as explained in Section 3.1, $g(x)$ has only real roots. Let t be the multiplicity of the zero-eigenvalue of the companion matrix B of $g(x)$. Then we have $i(B) = (\frac{1}{2}(\deg(g(x)) - t), \frac{1}{2}(\deg(g(x)) - t), t)$. It follows that $i(A)$ can be computed in $\mathbf{AC}^0(\mathbf{GapL})$. Actually, we can show a better upper bound for this problem. By $\textsc{Inertia}_{sym}$ we denote the restriction of $\textsc{Inertia}$, where the input matrix A is a symmetric integer matrix.

Theorem 5.1. *$\textsc{Inertia}_{sym}$ is in* $\mathbf{PL}$.

Proof. Let $A \in M_n$. If A is singular, we can compute $\chi_A(x)$ and determine the multiplicity t of eigenvalues 0 (in $\mathbf{GapL}$). Then it suffices to compute the inertia of the companion matrix of polynomial $\chi_A(x)/x^t$. The latter matrix is nonsingular. Therefore we may as well assume that A is nonsingular.

If A has no pair of opposite nonzero-eigenvalues, then $\textsc{Inertia}_{sym}$ is in $\mathbf{PL}$, as explained in Section 3.2. Therefore we consider the case when A has some pair of opposite nonzero eigenvalues. The idea is to determine a positive rational number ε such that the value $i_+(M)$ of the matrix $M = A - \varepsilon I$ is equal to $i_+(A)$ and the Routh-Hurwitz matrix $\Omega(M)$ is regular. Then we can apply Theorem 3.3 to compute $i_+(M)$.

The *spectral radius* of A is a bound on the distance of the eigenvalues of A from the origin. Furthermore, if $\| \cdot \|$ is any matrix norm, then $\rho(A) \leq \| A \|$ (see [HJ85]). We choose $\| \cdot \|$ as the *maximum column sum matrix norm*, i.e., $\nu(A) = \| A \| = \max_{1 \leq j \leq n} \sum_{i=1}^{n} |a_{ij}|$, and we have $\rho(A) \leq \nu(A)$. Because A is nonsingular, we can define $r_1 = (\nu(A^{-1}) + 1)^{-1}$. Let $\lambda_1(A), \ldots, \lambda_n(A)$ be the eigenvalues of A. Then $\lambda_i(A) > r_1$, for $i = 1, \ldots, n$. Now let $0 < \varepsilon \leq r_1$ and define matrix $M = A - \varepsilon I$. The eigenvalues of M are $\lambda_1(A) - \varepsilon, \ldots, \lambda_n(A) - \varepsilon$ and we have $i_+(M) = i_+(A)$.

It remains to determine ε such that $\Omega(M)$ is nonsingular. Observe that M has this property iff for all $i \neq j$

$$\lambda_i(A) - \varepsilon \neq -(\lambda_j(A) - \varepsilon) \Longleftrightarrow \quad \lambda_i(A) + \lambda_j(A) \neq 2\varepsilon. \tag{5}$$

The eigenvalues of $S = A \oplus A$ are $\lambda_i(A) + \lambda_j(A)$ for all $1 \leq i, j \leq n$. Thus equivalent to condition on the right-hand side of (5) is that 2ε is not an eigenvalue of S. Matrix S is singular, because A has some pair of opposite nonzero-eigenvalues. If we decompose $\chi_S(x) = x^k \chi_S^*(x)$ such that $\chi_S^*(0) \neq 0$, then the companion matrix S^* of $\chi_S^*(x)$ is nonsingular and k is exactly the multiplicity of the eigenvalue 0 of S. Since S^* is nonsingular, we can define $r_2 = (\nu({S^*}^{-1}) + 1)^{-1}$. Each of the eigenvalues of S^* has absolute value greater than r_2. Hence $\Omega(M)$ is nonsingular if $0 < 2\varepsilon \leq r_2$. In summary, we can choose $\varepsilon = \min\{r_1, r_2/2\}$.

The value ε, each element of M, and each of the Routh-Hurwitz determinants of $\Omega(M)$ can be computed in $\mathbf{GapL}$, because the elements of A^{-1} and ${S^*}^{-1}$ are computable in $\mathbf{GapL}$ ([AAM99]). Therefore $\textsc{Inertia}_{sym}$ is in $\mathbf{PL}$ by Theorem 3.3. □

Since each bit of $i_+(A)$ can be verified in **PL**, the values of $i(A)$ can be verified in **PL**, too. This implies that the problem of testing whether two given symmetric matrices have the same inertia (that is CONGRUENCE) is in **PL**. This solves an open problem in [HT00].

Corollary 5.2. CONGRUENCE*,* POSDEFINITE*,* POSSEMIDEFINITE $\in$ **PL**.

Summary and Open Questions

The table summarizes the lower and upper bounds for some of the problems considered in this paper. An obvious task for further research is to close the gap between the lower and the upper bound where it doesn't match.
A major challenge remains to compute the inertia in general.

Problem	hard for	contained in
POSSTABLE	**PL**	**PL**
POSSEMISTABLE	**PL**	$\mathbf{AC}^0(\mathbf{GapL})$
CONGRUENCE	$\mathbf{AC}^0(\mathbf{C_=L})$	**PL**
POSDEFINITE	$\mathbf{coC_=L}$	**PL**

Acknowledgments

We thank Eric Allender and the referees for many helpful comments on the paper. In particular, Eric pointed us to the results in [AAM99] from which we got that all Routh-Hurwitz determinants are computable in **GapL**.

References

AAM99. E. Allender, V Arvind, and M. Mahajan. Arithmetic complexity, Kleene closure, and formal power series, 1999.

ABO. E. Allender and M. Ben-Or. Electronic Communication, 2001.

ABO99. E. Allender, R. Beals, and M. Ogihara. The complexity of matrix rank and feasible systems of linear equations. *Computational Complexity*, 8:99–126, 1999.

All02. E. Allender. Personal Communication, 2002.

AO96. E. Allender and M. Ogihara. Relationship among PL, #L, and the determinant. *RAIRO-Theoretical Informatics and Applications*, 30:1–21, 1996.

Ber84. S. Berkowitz. On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18:147–150, 1984.

Dam91. C. Damm. DET = $L^{(\#L)}$. Technical Report Informatik-Preprint 8, Fachbereich Informatik der Humboldt Universitaet zu Berlin, 1991.

FFK94. S. Fenner, L. Fortnow, and S. Kurtz. Gap-definable counting classes. *Journal of Computer and System Sciences*, 48:116–148, 1994.

For00. S. Fortune. Exact computation of the inertia of symmetric integer matrices. In *32th Symposium on Theory of Computing, STOC 2000*, pages 556–564. ACM Press, 2000.

Gan77. F. Gantmacher. *The Theory of Matrices*, volume 1 and 2. AMS Chelsea Publishing, 1977.

Gra81. A. Graham. *Kronnecker Products and Matrix Calculus: With Applications.* Ellis Horwood Ltd., 1981.

HJ85. R. Horn and C. Johnson. *Matrix Analysis.* Cambridge University Press, 1985.

HJ91. R. Horn and C. Johnson. *Topics in Matrix Analysis.* Cambridge University Press, 1991.

HT00. T. M. Hoang and T. Thierauf. The complexity of verifying the characteristic polynomial and testing similarity. In *15th IEEE Conference on Computational Complexity (CCC)*, pages 87–95. IEEE Computer Society Press, 2000.

HT01. T. M. Hoang and T. Thierauf. The complexity of the minimal polynomial. In *26th International Symposium, MFCS 2001*, pages 408–420. Springer, 2001.

HT02. T. M. Hoang and T. Thierauf. The complexity of the characteristic and the minimal polynomial. Invited paper to the special issue in *Theoretical Computer Science* of the 26th MFCS conference 2001, to appear, 2002.

Koz91. D. Kozen. *The Design and Analysis of Algorithms.* Springer-Verlag, 1991.

Nef94. C. A. Neff. Specified precision root isolation is in NC. *Journal of Computer and System Science*, 48:429–463, 1994.

NR96. C. A. Neff and J. H. Reif. An efficient algorithm for the complex roots problem. *Journal of Complexity*, 12:81–115, 1996.

Ogi98. M. Ogihara. The PL hierarchy collapses. *SIAM Journal on Computing*, 27:1430–1437, 1998.

ST98. M. Santha and S. Tan. Verifying the determinant in parallel. *Computational Complexity*, 7:128–151, 1998.

Tod91. S. Toda. Counting problems computationally equivalent to the determinant. Technical Report CSIM 91-07, Dept. of Computer Science and Information Mathematics, University of Electro-Communications, Chofu-shi, Tokyo 182, Japan, 1991.

Val79a. L. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.

Val79b. L. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8:410–421, 1979.

Val92. L. Valiant. Why is Boolean complexity theory difficult. In M.S. Paterson, editor, *Boolean Function Complexity*, London Mathematical Society Lecture Notes Series 169. Cambridge University Press, 1992.

Vin91. V Vinay. Counting auxiliary pushdown automata and semi-unbounded arithmetic circuits. In *6th IEEE Conference on Structure in Complexity Theory*, pages 270–284, 1991.

The Quantum Communication Complexity of the Pointer Chasing Problem: The Bit Version

Rahul Jain[1], Jaikumar Radhakrishnan[1], and Pranab Sen[2,⋆]

[1] STCS, Tata Institute of Fundamental Research, Mumbai 400005, India
{rahulj,jaikumar}@tcs.tifr.res.in
[2] LRI, UMR 8623, Université de Paris–Sud, 91405 Orsay, France
pranab@lri.fr

Abstract. We consider the two-party quantum communication complexity of the bit version of the pointer chasing problem when the 'wrong' player starts, originally studied by Klauck, Nayak, Ta-Shma and Zuckerman [7]. We show that in any quantum protocol for this problem, the two players must exchange $\Omega(n/k^4)$ qubits. This improves the previous best lower bound of $\Omega\left(\frac{n}{2^{2^{O(k)}}}\right)$ in [7], and comes significantly closer to the best upper bounds known: $O(n + k \log n)$ (classical deterministic [12]) and $O(k \log n + \frac{n}{k}(\log^{(\lceil k/2 \rceil)} n + \log k))$ (classical randomised [7]). Our result demonstrates a separation between the communication complexity of k and $k-1$ round bounded error quantum protocols, for all $k < O((m/\log^2 m)^{1/5})$, where m is the size of the inputs to Alice and Bob. Earlier works could prove such a separation for much smaller k only. Our proof uses a round elimination argument for a class of quantum sampling protocols with correlated input generation, making better use of information-theoretic tools than previous works.

1 Introduction

We consider the following problem in the two-party communication model.

> Let V_A and V_B be disjoint sets of size n. Alice is given a function $F_A : V_A \to V_B$ and Bob is given a function $F_B : V_B \to V_A$. Let $F \stackrel{\Delta}{=} F_A \cup F_B$. There is a fixed vertex s in V_B. The players exchange messages, and the last recipient of a message needs to determine the least significant bit of $F^{(k+1)}(s)$ (denoted by $\mathrm{lsb}(F^{(k+1)}(s))$), where k and s are known to both parties in advance.

If Bob starts the communication, there is a straightforward classical deterministic protocol where one of the players can determine the answer after k messages of $\log n$ bits have been exchanged. It appears much harder, however, to solve the problem efficiently with k messages, when Alice is required to send the first message. We refer to this as the *pointer chasing* problem P_k.

⋆ Supported by the EU 5th framework project QAIP IST-1999-11234, and by CNRS/STIC 01N80/0502 and 01N80/0607 grants.

Background: The pointer chasing problem has been extensively studied in the past to show rounds versus communication tradeoffs in classical communication complexity. Nisan and Wigderson [10] showed (following some earlier results of Papadimitriou and Sipser [11], and Duris, Galil and Schnitger [4]) that the players must exchange $\Omega(\frac{n}{k^2} - k \log n)$ bits to solve P_k; their bound was improved by Klauck [6] to $\Omega(\frac{n}{k} + k)$. These lower bounds are for randomised protocols. For deterministic protocols, Nisan and Wigderson showed an $\Omega(n)$ lower bound. A deterministic protocol for P_k with $O(n + k \log n)$ bits of communication was given by Ponzio, Radhakrishnan and Venkatesh [12], and a randomised protocol for P_k with $O(k \log n + \frac{n}{k}(\log^{(\lceil k/2 \rceil)} n + \log k))$ bits of communication was given by Klauck, Nayak, Ta-Shma and Zuckerman [7]. Thus, the lower and upper bounds for P_k are quite close in the classical setting.

In the quantum setting, the problem P_k has been studied recently by Klauck, Nayak, Ta-Shma and Zuckerman [7], who, using interesting information-theoretic techniques, showed a lower bound of $\Omega(\frac{n}{2^{2^{O(k)}}})$ on its quantum communication complexity. This bound deteriorates rapidly with k, and becomes trivial for $k = \Omega(\log \log n)$. We improve this lower bound.

Result 1 *In any bounded error quantum communication protocol for P_k, Alice and Bob must exchange $\Omega(\frac{n}{k^4})$ qubits.*

This result demonstrates a separation between the communication complexity of k and $k-1$ round bounded error quantum protocols, for all $k < O((m/\log^2 m)^{1/5})$, where m is the size of the inputs to Alice and Bob. Earlier works could prove such a separation for much smaller k only.

Our Proof Technique: The underlying information-theoretic tools we use are, in fact, mainly taken from the paper of Klauck, Nayak, Ta-Shma and Zuckerman [7]. Our proofs use the *round elimination* method, stated explicitly in the classical communication complexity setting by Miltersen, Nisan, Safra and Wigderson [8]. This technique was applied in the quantum setting by Klauck *et al.* [7], who developed several quantum information-theoretic tools, notably the *average encoding theorem* and the *local transition theorem*. Their arguments were refined further by Sen and Venkatesh [14]. Recently, Jain, Radhakrishnan and Sen [5] showed an optimal $\Omega(n \log^{(k)} n)$ lower bound for the *full version* of the pointer chasing problem, where the full description of $F^{(k+1)}(s)$ is required, and not just its least significant bit. This lower bound was also obtained by a round elimination argument, using a technical quantum information-theoretic tool — the *Substate Theorem*. In this paper, we adapt their argument to the *bit version* P_k of the problem. For this, we consider a slightly different pointer chasing problem Q_k, and quantum protocols for Q_k where the two players generate their own inputs in a correlated fashion and then proceed to compute the answer. In other words, we consider a special class of quantum sampling protocols for Q_k. It is known that in general, the quantum sampling complexity of a function can be exponentially smaller than its quantum communication complexity [2,13]. So, to keep our problem non-trivial, we must impose some restrictions on the way Alice and Bob behave. First, we insist that the inputs they generate must be sufficiently rich. Second, the amount of communication before the input is generated, is limited. In previous round elimination arguments, the inputs were supplied to the two players from 'outside'. While this worked well for many problems, for the pointer chasing problem it made things difficult. However, letting Alice and Bob generate their own inputs gives rise to new technical difficulties. This is because the inputs they generate are not exactly what we want, but only close to it. So, we need to apply an input correction step that converts a protocol whose inputs have a probability distribution close to the one we desire, into one where the inputs have exactly the probability distribution we want. Our proof does not use the Substate theorem of Jain *et al.*[5]; rather, it uses the earlier quantum information theoretic tools of Klauck *et al.* [7]. Overall, we believe, the main contribution of this work is in showing how existing information-theoretic tools can be better exploited for round elimination in quantum communication protocols.

1.1 Organisation of the Rest of the Paper

In the next section, we derive our lower bound for P_k assuming a round elimination lemma. In Section 3, we collect the information-theoretic tools that are required for the proof of the round elimination lemma. Finally, in Section 4, we describe the round elimination argument in detail.

2 Lower Bound for the Pointer Chasing Problem P_k

In this section, we prove our lower bound for P_k by proving, in fact, a lower bound for a related problem Q_k. The lower bound for Q_k is proved assuming a round elimination lemma. The round elimination lemma will be proved in a later section.

2.1 Quantum Communication Protocols

We consider two party quantum communication protocols as defined by Yao [15]. From now on, the term *measurement* means a von Neumann measurement in the computational basis, unless indicated otherwise.

Definition 1 (Safe transformation, protocols). *Let $\mathcal{H}$ and $\mathcal{K}$ be finite-dimensional Hilbert spaces, with computational orthonormal bases $(|h\rangle : h \in H)$ and $(|k\rangle : k \in K)$. We say that a unitary transformation U on $\mathcal{H} \otimes \mathcal{K}$* acts safely *on $\mathcal{H}$ if there exist unitary transformations $(U_h : h \in H)$ acting on $\mathcal{K}$ such that for all $h \in H$ and $k \in K$,*

$$U : |h\rangle \otimes |k\rangle \mapsto |h\rangle \otimes U_h|k\rangle.$$

We say that a quantum communication protocol acts safely on a register R, if all unitary transformations in the protocol act safely on R, and R is never sent as part of a message. We say that a protocol is safe *if Alice and Bob act safely on their input registers.*

When the inputs to a quantum communication protocol are classical, we can always assume that the protocol is safe. This is because the inputs to Alice and Bob are in computational basis states, and hence, the players can make a secure copy of their inputs before beginning the protocol.

In the problem P_k, we assume, without loss of generality, that the number of vertices $n = 2^r$ for some $r \geq 1$. Thus, the vertices in V_A and V_B have binary encodings of length $\log n$. Alice and Bob exchange messages $M_1, \dots, M_k$, having lengths $c_1 n, \dots, c_k n$, via a safe quantum protocol in order to determine $\mathrm{lsb}(F^{(k+1)}(s))$. Alice starts the communication, that is, she sends M_1. The player receiving M_k places a qubit in a special register Ans. We require that the bit obtained by measuring Ans should be the correct answer (i.e. equal to $\mathrm{lsb}(F^{(k+1)}(s))$), with probability at least $3/4$, for all F_A, F_B.

We prove our lower bound for P_k using an inductive argument. It will be convenient to state our induction hypothesis by means of predicates Q_k^A and Q_k^B, defined below. Roughly, the induction proceeds as follows. We show that if there is an efficient protocol for P_k, then Q_k^A is true. We then show independently that Q_ℓ^A implies $Q_{\ell-1}^B$ and Q_ℓ^B implies $Q_{\ell-1}^A$, and that Q_0^A and Q_0^B are false. Thus, there is no efficient protocol for P_k.

We first define $Q_k^A(c_1, \dots, c_k, n_a, n_b, \epsilon)$ for $k \geq 1$. Then, separately, we define $Q_0^A(\epsilon)$. For $k \geq 0$, Q_k^B is the same as Q_k^A, with the roles of Alice and Bob reversed. Consequently, all our statements involving Q_k^A and Q_k^B have two forms, where one is obtained from the other by reversing the roles of Alice and Bob. We will typically state just one of them, and let the reader infer the other.

2.2 The Predicate $Q_k^A, k \geq 1$

The predicate $Q_k^A(c_1, \dots, c_k, n_a, n_b, \epsilon)$ holds if there is a quantum protocol $\mathcal{P}$ of the following form.

Input generation: In $\mathcal{P}$, Alice and Bob 'generate' most of their inputs themselves. Alice has n input registers $(F_A[u] : u \in V_A)$ and Bob has n input registers $(F_B[v] : v \in V_B)$. There is a fixed vertex $s \in V_B$, that is known to both players. Each of Alice's registers has $\log n$ qubits so that it can hold a description of a vertex in V_B; similarly, each of Bob's registers can hold a description of a vertex in V_A. In addition, Alice and Bob have registers for their 'work' qubits W_A and W_B.

When $\mathcal{P}$ starts, Alice's registers are all initialised to zero. On Bob's side, the register $F_B[s]$ starts off with the uniform superposition $\frac{1}{\sqrt{n}} \sum_{a \in V_A} |a\rangle$; his other registers are all zero.

Alice starts by generating a pure state in $\widetilde{M_1} M_1$, where $\widetilde{M_1}, M_1$ are each $c_1 n$-qubit registers. Then she applies a unitary transformation U_A on $\widetilde{M_1}$ plus some ancilla qubits initialised to zero, to generate a suitable pure state in registers F_A, W_A and M_1 (note that the qubits of $\widetilde{M_1}$ plus the above ancilla qubits constitute the qubits of F_A, W_A). After the application of U_A, F_A holds the 'generated input' to Alice for the pointer chasing problem, and W_A holds Alice's 'work qubits'. Alice then sends M_1 to Bob.

Now, Bob generates his input by applying a unitary transformation U_B on the registers M_1, F_B and W_B. U_B operates safely on $F_B[s]$. After the application of U_B, F_B holds the 'generated input' to Bob for the pointer chasing problem, and W_B holds Bob's 'work qubits'.

We will use F_A, F_B to refer to the actual states of the respective registers; f_A, f_B will denote the states that would result, were we to measure F_A, F_B. Thus, typically F_A, F_B will be parts of a pure state (the global state of Alice's and Bob's qubits), whereas f_A, f_B will be mixtures of computational basis states.

For our predicate $Q_k^A(c_1, \ldots, c_k, n_a, n_b, \epsilon)$ to hold for a $k > 0$, this input generation process must satisfy some conditions.
Requirement 1: The random variables $(f_A[u] : u \in V_A)$ are independent. The same is true for $(f_B[v] : v \in V_B)$ too. However, there might be dependence between f_A and f_B.
Requirement 2: There is a subset $X_A \subseteq V_A$ of size at most n_a such that the random variables $(f_A[u] : u \in X_A)$ are set to constants, and for $u \in V_A - X_A$, $f_A[u]$ is uniformly distributed.
Requirement 3: There is a subset $X_B \subseteq V_B - \{s\}$ of size at most n_b such that the random variables $(f_B[v] : v \in X_B)$ are set to constants, and for $v \in V_B - X_B$, $f_B[v]$ is uniformly distributed. Note that $f_B[s]$ is automatically uniformly distributed, because initially $F_B[s]$ contains the uniform superposition, and U_B acts safely on $F_B[s]$.
Communication: After U_A, U_B have been applied, Alice and Bob follow a quantum protocol exchanging further messages $M_2, \ldots, M_k$ of lengths $c_2 n, \ldots, c_k n$. Bob sends the message M_2. The rest of the protocol $\mathcal{P}$ (after the application of U_A and U_B) is required to act safely on registers F_A, F_B. At the end of $\mathcal{P}$, the player who receives M_k places a qubit in a special register Ans. $\mathcal{P}$ then terminates.
The error probability: Once $\mathcal{P}$ has terminated, all registers are measured. Let ans denote the value observed in Ans, and let f_A and f_B be the values observed in F_A and F_B; we treat f_A and f_B as functions from V_A to V_B and V_B to V_A respectively. Let $f \triangleq f_A \cup f_B$.
Requirement 4: $\Pr[\mathsf{ans} \neq \mathrm{lsb}(f^{(k+1)}(s))] \leq \epsilon$.

2.3 The Predicate Q_0^A

The predicate $Q_0^A(\epsilon)$ holds if there is a quantum protocol $\mathcal{P}$ of the following form. At the start of $\mathcal{P}$, Alice's registers are all initialised to zero. On Bob's side, the register $F_B[s]$ starts off with the uniform superposition $\frac{1}{\sqrt{n}} \sum_{a \in V_A} |a\rangle$; his other registers are all zero.

Alice starts by generating a pure state in $\widetilde{M_1} M_1$, where $\widetilde{M_1}, M_1$ are each $c_1 n$-qubit registers. Then she applies a unitary transformation U_A on $\widetilde{M_1}$ plus some ancilla qubits initialised to zero to generate a suitable pure state in registers F_A, W_A and M_1. Alice then sends M_1 to Bob.

Now, Bob generates his input by applying a unitary transformation U_B on the registers M_1, F_B and W_B. U_B operates safely on $F_B[s]$.

Alice now places a qubit in a special register Ans and $\mathcal{P}$ terminates. All the registers of Alice and Bob are measured. We require that $\mathsf{ans} \neq \mathrm{lsb}(f_B[s])$ with probability at most ϵ.

2.4 Tying Things up

We now state some basic lemmas which will be useful in proving the desired lower bound for P_k. The proofs are omitted due to lack of space.

Lemma 1. *If $Q_0^A(\epsilon)$ is true then $\epsilon \geq \frac{1}{2}$.*

Lemma 2. *If there is a safe quantum protocol for P_k with messages of lengths $c_1 n, \ldots, c_k n$, and worst case error at most $1/4$, then $Q_k^A(c_1, \ldots, c_k, 0, 0, 1/4)$ is true.*

Lemma 3 (Round elimination). *(a) For $k \geq 2$, if $Q_k^A(c_1, \ldots, c_k, n_A, n_B, \epsilon)$ holds with $n_a < n$ then $Q_{k-1}^B(c_1 + c_2, c_3, \ldots, c_k, n_a, n_b + 1, \epsilon')$ holds with $\epsilon' \overset{\Delta}{=} \left(\frac{n}{n-n_a}\right)\left(\epsilon + 2((2\ln 2)c_1)^{1/4}\right)$.*
(b) If $Q_1^A(c_1, n_a, n_b, \epsilon)$ holds (with $n_a < n$), then $Q_0^A(\epsilon')$ holds, where ϵ' is the same as in (a).

The next section is devoted to the proof of this lemma. Now, we assume this lemma and prove our main lower bound.

Theorem 1. *Suppose $k \leq O(n^{1/4})$ and $Q_k^A(c_1, \ldots, c_k, 0, 0, 1/4)$ holds. Then, $c_1 + \cdots + c_k = \Omega(k^{-4})$.*

Now, by using Lemma 2, we can derive our lower bound for P_k.

Corollary 1 (Main result). *In any bounded error quantum protocol for P_k, Alice and Bob must exchange a total of $\Omega(\frac{n}{k^4})$ qubits.*

3 Preliminaries

We now recall some basic definitions and facts from probability and information theory, which will be useful in proving our round elimination lemma. For excellent introductions to classical and quantum information theory, see the books by Cover and Thomas [3] and Nielsen and Chuang [9] respectively.

If A is a quantum system with density matrix ρ, then $S(A) \overset{\Delta}{=} S(\rho) \overset{\Delta}{=} -\mathrm{Tr}\, \rho \log \rho$ is the *von Neumann entropy* of A. If A, B are two disjoint quantum systems, the *mutual information* of A and B is defined as $I(A : B) \overset{\Delta}{=} S(A) + S(B) - S(AB)$.

Fact 1 (see e.g. [7]) *Let $X_1, \ldots, X_n$ be independent classical random variables and let M be a quantum encoding of $X \overset{\Delta}{=} X_1 \ldots X_n$. Then, $I(X : M) \geq \sum_{i=1}^{n} I(X_i : M)$. Also, if M is n qubits long, then $I(X : M) \leq n$.*

We shall work with various measures of distance between classical and quantum states. For probability distributions D and D' on a finite set X, their *total variational distance* is given by $\|D - D'\|_1 \stackrel{\Delta}{=} \sum_{x \in X} |D(x) - D'(x)|$. We will use the following elementary fact, which we state without proof.

Fact 2 *Suppose D, D' are two probability distributions on the same finite set X, whose total variation distance $\|D - D'\|_1$ is δ. Then, there exists a stochastic matrix $P \stackrel{\Delta}{=} (p_{xx'})_{x,x' \in X}$, such that $D = PD'$ and $\sum_{x' \in X} P(x', x')D(x') = 1 - \frac{\delta}{2}$. Let $\mathcal{H}$ be a Hilbert space with computational orthonormal basis $(|x\rangle : x \in X)$. Let C be a unitary transformation on $\mathcal{H} \otimes \mathcal{H}$ that maps computational basis vectors of the form $|x'\rangle \otimes |\mathbf{0}\rangle$ (where $\mathbf{0}$ is a special element of X) according to the rule*

$$|x'\rangle \otimes |\mathbf{0}\rangle \to |x'\rangle \otimes \sum_{x \in X} \sqrt{p_{xx'}} |x\rangle,$$

and maps other computational basis vectors suitably, preserving orthonormality. Let R, R' be registers that can hold states in $\mathcal{H}$, and R'' be a register of some fixed length. Let (R', R'') initially contain a superposition

$$|\phi\rangle_{R'R''} \stackrel{\Delta}{=} \sum_{x \in X} \sqrt{D'(x)}\, |x\rangle_{R'} \otimes |\phi_x\rangle_{R''},$$

where the subscripts denote registers, and for each $x \in X$, $|\phi_x\rangle$ is a pure state in R''. Let R initially contain the state $|0\rangle$. Suppose one were to apply C to (R', R), and then measure (R', R). Let the resulting random variables (taking values in X) be Z' and Z respectively. Then, Z' will have distribution D', Z will have distribution D and $\Pr[Z \neq Z'] \leq \frac{\delta}{2}$. Note that C acts safely on R'.

The *trace norm* of a linear operator A on a finite dimensional Hilbert space $\mathcal{H}$ is defined as $\|A\|_t \stackrel{\Delta}{=} \mathrm{Tr}\, \sqrt{A^\dagger A}$. The following fundamental fact shows that the *trace distance* between two density matrices ρ_1, ρ_2, $\|\rho_1 - \rho_2\|_t$, bounds how well one can distinguish between ρ_1, ρ_2 by a measurement.

Fact 3 (see e.g. [1]) *Let ρ_1, ρ_2 be two density matrices over the same finite dimensional Hilbert space. Let $\mathcal{M}$ be a general measurement (i.e. a POVM), and $\mathcal{M}\rho_i$ denote the probability distributions on the (classical) outcomes of $\mathcal{M}$ got by performing measurement $\mathcal{M}$ on ρ_i. Then $\|\mathcal{M}\rho_1 - \mathcal{M}\rho_2\|_1 \leq \|\rho_1 - \rho_2\|_t$.*

We will need the following 'average encoding theorem' of Klauck *et al.* [7]. Intuitively speaking, it says that if the mutual information between a classical random variable and its quantum encoding is small, then the various quantum 'codewords' are close to the 'average codeword'.

Fact 4 (Average encoding theorem [7]) *Suppose X, Q are two disjoint finite dimensional quantum systems, where X is a classical random variable, which takes value x with probability p_x, and Q is a quantum encoding $x \mapsto \sigma_x$ of X. Let the density matrix of the average encoding be $\sigma \stackrel{\Delta}{=} \sum_x p_x \sigma_x$. Then*

$$\sum_x p_x \|\sigma_x - \sigma\|_t \leq \sqrt{(2 \ln 2) I(X : Q)}.$$

We will also need the following 'local transition theorem' of Klauck *et al.* [7].

Fact 5 (Local transition theorem [7]) *Let ρ_1 and ρ_2 be two density matrices over the same finite dimensional Hilbert space $\mathcal{H}$, $\mathcal{K}$ any Hilbert space of dimension at least the dimension of $\mathcal{H}$, and $|\phi_i\rangle$ any purifications of ρ_i in $\mathcal{H} \otimes \mathcal{K}$. Then, there is a unitary transformation U on $\mathcal{K}$ that maps $|\phi_2\rangle$ to $|\phi_2'\rangle \overset{\Delta}{=} (I \otimes U)|\phi_2\rangle$ (I is the identity operator on $\mathcal{H}$), such that*

$$\||\phi_1\rangle\langle\phi_1| - |\phi_2'\rangle\langle\phi_2'|\|_t \le 2\sqrt{\|\rho_1 - \rho_2\|_t}.$$

4 Round Elimination: Proof of Lemma 3

We consider Part (a) first. Part (b) follows by using similar arguments, and we do not describe them explicitly. Suppose $Q_k^A(c_1, c_2, \ldots, c_k, n_a, n_b, \epsilon)$ is true and let protocol $\mathcal{P}$ satisfy the requirements. We will show that there is a protocol $\mathcal{Q}$ that satisfies the requirements for Q_{k-1}^B, with parameters stated in Lemma 3(a).

In what follows, subscripts of pure and mixed states will denote the registers which are in those states. For $u \in V_A$, we use the subscript u instead of $F_A[u]$. Similarly, for $v \in V_B$, we use the subscript v instead of $F_B[v]$. For example, we say that the register $F_B[s]$ is initially in the state $|\mu\rangle_s \overset{\Delta}{=} \frac{1}{\sqrt{n}} \sum_{u \in V_A} |u\rangle_s$. For $u \in V_A$, $F_{A,u}$ is a shorthand for registers $(F_A[w] : w \in V_A - \{u\})$. For $v \in V_B$, $F_{B,v}$ denotes likewise.

Suppose $a \in V_A$. Let T denote the registers M_1, $F_{A,a}$ and W_A. Let $|\psi^A_{a\to b}\rangle$ denote the (pure) state of T in protocol $\mathcal{P}$ just before Alice sends M_1 to Bob, if $f_A[a] = b$. (If $\Pr[f_A[a] = b] = 0$, then $|\psi^A_{a\to b}\rangle$ is defined to be the zero vector.) Let R denote the registers $F_{B,s}$ and W_B. Let $\ell_a \overset{\Delta}{=} 1$ if $a \in X_A$ and $\ell_a \overset{\Delta}{=} n$ otherwise. The global state vector of Alice and Bob in $\mathcal{P}$ just before Alice sends M_1 to Bob is

$$\frac{1}{\sqrt{n}} \sum_{a \in V_A} \frac{1}{\sqrt{\ell_a}} \sum_{b \in V_B} |b\rangle_a |\psi^A_{a\to b}\rangle_T |a\rangle_s |\mathbf{0}\rangle_R.$$

At this point in $\mathcal{P}$, the first message M_1 is sent to Bob. Let the rest of the protocol starting from the point just after M_1 is sent be $\mathcal{P}'$; that is, in $\mathcal{P}'$ Bob starts by generating his input from M_1, R and $F_B[s]$, sends the message M_2 to Alice, to which Alice responds with M_3, and so on. The error probability of $\mathcal{P}'$ is defined in a manner similar to that of $\mathcal{P}$.

Let $\epsilon_{a\to b}$ be the error probability when $\mathcal{P}'$ is run starting from the state $|b\rangle_a |\psi^A_{a\to b}\rangle_T |a\rangle_s |\mathbf{0}\rangle_R$. Since $\mathcal{P}'$ is safe, we have

$$\epsilon_{a\to b} = \Pr[\mathsf{ans} \ne f^{(k+1)}(s) \mid f_B[s] = a \text{ and } f_A[a] = b].$$

Also, we have

$$\epsilon = \mathop{\mathrm{E}}_{a,b}[\epsilon_{a\to b}] \ge \left(\frac{n - n_a}{n}\right) \mathop{\mathrm{E}}_{a \in V_A - X_A, b \in V_B}[\epsilon_{a\to b}]. \qquad (1)$$

In the first expectation, (a,b) are chosen with the same distribution as that of $(f_B[s], f_A[f_B[s]])$ of the given protocol $\mathcal{P}$; in the second, they are chosen uniformly and independently from the sets specified.

Let $\sigma_{1,a\to b}$ be the density matrix of register M_1 in the state $|\psi^A_{a\to b}\rangle$, and σ_1 be the density matrix of register M_1 in the state $\frac{1}{\sqrt{n}}\sum_{b\in V_B}|b\rangle_a|\psi^A_{a\to b}\rangle_T$. Note that $\sigma_1 = \frac{1}{n}\sum_{b\in V_B}\sigma_{1,a\to b}$, and σ_1 is independent of $a = f_B[s]$. Let $(M_1, \widetilde{M_1})$ contain the canonical purification of σ_1, where $\widetilde{M_1}$ is a $c_1 n$-qubit register. Let $\delta_{a\to b} \overset{\Delta}{=} \|\sigma_{1,a\to b} - \sigma_1\|_t$. Then by Fact 5, there exists a unitary transformation $U_{a\to b}$ that when applied to $\widetilde{M_1}$ together with ancilla qubits, takes the pure state $(M_1, \widetilde{M_1})$ to a pure state $|\tilde{\psi}^A_{a\to b}\rangle$ on registers F_A, W_A, M_1, such that

$$\left\| |b\rangle\langle b| \otimes |\psi^A_{a\to b}\rangle\langle\psi^A_{a\to b}| - |\tilde{\psi}^A_{a\to b}\rangle\langle\tilde{\psi}^A_{a\to b}| \right\|_t \le 2\sqrt{\delta_{a\to b}}. \tag{2}$$

In particular, if $\mathcal{P}'$ is started from the state $|\tilde{\psi}^A_{a\to b}\rangle_{F_A W_A M_1}|a\rangle_s|\mathbf{0}\rangle_R$ (instead of the state $|b\rangle_a|\psi^A_{a\to b}\rangle_T|a\rangle_s|\mathbf{0}\rangle_R$), the probability of error is at most $\epsilon_{a\to b} + \sqrt{\delta_{a\to b}}$, by Fact 3.

4.1 The Protocol $\mathcal{P}_{a\to b}$

Let us now fix $a \in V_A - X_A$ and $b \in V_B$ and consider the case when $f_B[s] = a$ and $f_A[a] = b$. We now describe a protocol $\mathcal{P}_{a\to b}$. This is just an intermediate protocol. Later, we will describe how we obtain our final protocol $\mathcal{Q}$ (satisfying the requirements of Q^B_{k-1}) from $\mathcal{P}_{a\to b}$. It will be helpful, meanwhile, to keep in mind that in $\mathcal{Q}$, the roles of Alice and Bob will be reversed, $F_B[s]$ will be fixed at $|a\rangle$ (thus, we will add s to X_B), a will be our new s, and the initial state of $F_A[a]$ will be the uniform superposition $\frac{1}{\sqrt{n}}\sum_{v\in V_B}|v\rangle$.

Step 1: Alice generates the canonical purification of σ_1 in registers $(M_1, \widetilde{M_1})$. Alice applies $U_{a\to b}$ to $\widetilde{M_1}$ plus some ancilla qubits initialised to zero. She sends M_1 to Bob.
Step 2: Bob sets $F_B[s]$ to $|a\rangle$, and the register R to $|0\rangle$.
Step 3: Alice and Bob now proceed according to the protocol $\mathcal{P}'$.

Remark on the Inputs Generated: Let $\tilde{f}_{A,a\to b}$ be the random variable with distribution $\tilde{D}_{a\to b}$, resulting on measuring F_A in the state $|\tilde{\psi}^A_{a\to b}\rangle$. Let $f_{A,a\to b}$ be the random variable with distribution $D_{a\to b}$, resulting on measuring F_A in the state $|b\rangle_a|\psi^A_{a\to b}\rangle_T$; that is, $D_{a\to b}$ is the distribution of f_A in the original protocol $\mathcal{P}$ conditioned on the event $f_A[a] = b$. Then, it follows from (2) and Fact 3 that

$$\|D_{a\to b} - \tilde{D}_{a\to b}\|_1 \le 2\sqrt{\delta_{a\to b}}. \tag{3}$$

Let $\tilde{\epsilon}_{a\to b}$ denote the error probability of $\mathcal{P}_{a\to b}$. Then $\tilde{\epsilon}_{a\to b} \le \epsilon_{a\to b} + \sqrt{\delta_{a\to b}}$. In $\mathcal{P}_{a\to b}$, f_B satisfies Requirements 1 and 2 with respect to $X_B \cup \{s\}$ (note that we want to eventually switch the roles of Alice and Bob). However, we cannot say that f_A satisfies Requirements 1 and 3. To get over this hurdle, we have to do a "correction process" on Alice's input registers as described below, leading us to protocol $\mathcal{P}'_{a\to b}$.

4.2 The Protocol $\mathcal{P}'_{a\to b}$

We now describe the protocol $\mathcal{P}'_{a\to b}$. In $\mathcal{P}'_{a\to b}$, in addition to her registers in $\mathcal{P}_{a\to b}$, Alice has a fresh set of registers $\hat{F}_A$ of $n \log n$ qubits, initialised to zero. After $\mathcal{P}'_{a\to b}$ terminates, $\hat{F}_A$ is treated as Alice's input register while determining whether $\mathcal{P}'_{a\to b}$ succeeded or not.

Step 1(a): Alice generates the canonical purification of σ_1 in registers $(M_1, \widetilde{M_1})$. Alice applies $U_{a\to b}$ to $\widetilde{M_1}$ plus some ancilla qubits initialised to zero.

Step 1(b): Alice now does a 'correction process' on her input registers in order to satisfy Requirements 1 and 3. Let $C_{a\to b}$ be the unitary transformation 'correcting' $\tilde{D}_{a\to b}$ to $D_{a\to b}$ according to Fact 2. Alice applies $C_{a\to b}$ to registers $F_A, \hat{F}_A$. Note that $C_{a\to b}$ is safe on F_A. The input generation for Alice is now complete.

Step 1(c): Alice sends M_1 to Bob.

Step 2: Bob sets $F_B[s]$ to $|a\rangle$, and the register R to $|0\rangle$.

Step 3: From this point on, Alice and Bob just follow $\mathcal{P}'$. The registers $\hat{F}_A$ are not 'touched' by $\mathcal{P}'$.

While executing $\mathcal{P}'$, Alice's old input registers (in protocol $\mathcal{P}$) F_A are used. Alice's new input registers $\hat{F}_A$ are not touched by any unitary transformation in $\mathcal{P}'$. Also, $\mathcal{P}'$ is safe on F_A. At the end of $\mathcal{P}'_{a\to b}$ however, we will check the correctness of the answer with respect to $\hat{F}_A$ and F_B. Let $\hat{f}_{A,a\to b}$ denote the random variable got by measuring $\hat{F}_A$ at the end of $\mathcal{P}'_{a\to b}$. Note that if we measure $(F_A, \hat{F}_A)$ at the end of $\mathcal{P}'_{a\to b}$, the resulting random variables $(\tilde{f}_{A,a\to b}, \hat{f}_{A,a\to b})$ have distribution precisely $\tilde{D}_{a\to b}$ and $D_{a\to b}$, and

$$\Pr[\tilde{f}_{A,a\to b} \neq \hat{f}_{A,a\to b}] \leq \frac{1}{2} \cdot (2\sqrt{\delta_{a\to b}}) = \sqrt{\delta_{a\to b}}.$$

Let $\epsilon'_{a\to b}$ be the error probability of $\mathcal{P}'_{a\to b}$. Then one can see that

$$\epsilon'_{a\to b} \leq \Pr[\tilde{f}_{A,a\to b} \neq \hat{f}_{A,a\to b}] + \tilde{\epsilon}_{a\to b} \leq \epsilon_{a\to b} + 2\sqrt{\delta_{a\to b}}. \tag{4}$$

4.3 The Protocol $\mathcal{P}'_a$

In $\mathcal{P}'_{a\to b}$, $\hat{f}_A[a] = b$ and $f_B[s] = a$. We now describe a protocol $\mathcal{P}'_a$ where $F_B[s]$ is fixed to $|a\rangle$, but $\hat{F}_A[a]$ contains the uniform superposition $\frac{1}{\sqrt{n}} \sum_{v \in V_B} |v\rangle$ initially. In $\mathcal{P}'_a$, Bob starts the communication and $k-1$ rounds take place. $\hat{F}_A, F_B$ are Alice's and Bob's input registers respectively in $\mathcal{P}'_a$. Alice's old (in $\mathcal{P}$) input registers F_A continue to exist, but they count as her work qubits now. In $\mathcal{P}'_a$, in addition to her registers in $\mathcal{P}'_{a\to b}$, Alice has a fresh register Z of $\log n$ qubits. Initially, all qubits are initialised to zero.

Step 1: Bob generates the canonical purification of σ_1 in registers $(M_1, \widetilde{M_1})$. He sets $F_B[s]$ to $|a\rangle$, and using the transformation U_B, generates his inputs F_B and work qubits W_B. Then he generates the first message M'_1 of protocol $\mathcal{P}'$ (this corresponds to message M_2 of $\mathcal{P}$), and sends M'_1 along with $\widetilde{M_1}$ to Alice.

Step 2(a): Alice places the uniform superposition state $\frac{1}{\sqrt{n}}\sum_{v\in V_B}|v\rangle$ in register $\hat{F}_A[a]$. On receiving $\widetilde{M}_1$, Alice applies a unitary transformation on $\widetilde{M}_1$ plus some ancilla qubits initialised to zero. This unitary transformation is nothing but $U_{a\to b}$ if $\hat{f}_A[a]=b$. Note that it is safe on $\hat{F}_A[a]$, and it makes $z=\hat{f}_A[a]$.
Step 2(b): Alice does an 'input correction process' as follows. Let Y denote the registers $\hat{F}_{A,a}$ and Z. Alice applies $C_{a\to b}$ to registers F_A, Y if $\hat{f}_A[a]=b$. Note that this 'input correction process' is safe on $\hat{F}_A[a]$.
Step 3: Alice resumes the protocol $\mathcal{P}'$. Note that Bob has already executed the first step of $\mathcal{P}'$ and sent M_1'. Alice responds to M_1' as before. $\mathcal{P}'$ does not 'touch' $\hat{F}_A, Z$.

Let $\hat{\epsilon}_a'$ be the error probability of $\mathcal{P}_a'$. Then using (4), we get

$$\hat{\epsilon}_a' = \mathop{\mathrm{E}}_{b\in V_B}[\epsilon_{a\to b}'] \le \mathop{\mathrm{E}}_{b\in V_B}[\epsilon_{a\to b}] + 2((2\ln 2)I(f_A[a]:M_1))^{1/4}, \tag{5}$$

where b is chosen uniformly from V_B in the expectations above, and $I(f_A[a]:M_1)$ is mutual information between $f_A[a]$ and M_1 in the original protocol $\mathcal{P}$. The inequality follows from Fact 4, using linearity of expectation and concavity of square root.

4.4 The Final Protocol $\mathcal{Q}$

We first describe a protocol $\mathcal{P}_a$ with $k-1$ rounds of communication and Bob starting, which satisfies Requirements 1, 2 and 3 with respect to X_A and $X_B \cup \{s\}$, the roles of Alice and Bob being reversed, with a as the new 'starting vertex'. The only difference between $\mathcal{P}_a$ and $\mathcal{P}_a'$ is in Step 1. Let $\widehat{M}_1$ denote the first message of $\mathcal{P}_a'$, that is, $\widehat{M}_1 = (\widetilde{M}_1, M_1')$. $\widehat{M}_1$ is $(c_1+c_2)n$-qubits long. Let $(\widetilde{\widehat{M}}_1, \widehat{M}_1)$ contain a canonical purification of $\widehat{M}_1$, where $\widetilde{\widehat{M}}_1$ is $(c_1+c_2)n$-qubits long. In Step 1 of $\mathcal{P}_a$, Bob applies an appropriate unitary transformation on $\widetilde{\widehat{M}}_1$ plus some ancilla qubits initialised to zero, to generate the same state as in at the end of Step 1 of $\mathcal{P}_a'$. After this, $\mathcal{P}_a$ proceeds in the same fashion as $\mathcal{P}_a'$. The success probability $\hat{\epsilon}_a$ of $\mathcal{P}_a$ is the same as the success probability ϵ_a' of $\mathcal{P}_a'$.

Using Fact 1, we see that $\mathrm{E}_{a\in V_A - X_A}[I(f_A[a]:M_1)] \le \frac{nc_1}{n-n_a}$. From this and (1), (5) and concavity of the fourth root function, we get that

$$\begin{aligned}
\mathop{\mathrm{E}}_{a\in V_A-X_A}[\hat{\epsilon}_a] &\le \mathop{\mathrm{E}}_{a\in V_A-X_A, b\in V_B}[\epsilon_{a\to b}] + 2((2\ln 2)\mathop{\mathrm{E}}_{a\in V_A-X_A}[I(f_A[a]:M_1)])^{1/4} \\
&\le \frac{n\epsilon}{n-n_a} + 2\left(\frac{(2\ln 2)nc_1}{n-n_a}\right)^{1/4} \\
&\le \left(\frac{n}{n-n_a}\right)\left(\epsilon + 2((2\ln 2)c_1)^{1/4}\right) \stackrel{\Delta}{=} \epsilon'.
\end{aligned}$$

In the expectations above, a and b are chosen uniformly and independently from the sets specified.

Thus, there exists an $a\in V_A - X_A$ such that $\hat{\epsilon}_a \le \epsilon'$. Our final protocol $\mathcal{Q}$ is nothing but $\mathcal{P}_a$ for this a. It can be verified that $\mathcal{Q}$ satisfies the requirements for $Q_{k-1}^B(c_1+c_2, c_3, \ldots, c_k, n_a, n_b+1, \epsilon')$. This completes the proof of Lemma 3.

Acknowledgements

We thank Hartmut Klauck for going through an earlier draft of the paper and providing us useful feedback.

References

1. D. Aharonov, A. Kitaev, and N. Nisan. Quantum circuits with mixed states. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pages 20–30, 1998. Also quant-ph/9806029.
2. A. Ambainis, L. Schulman, A. Ta-Shma, U. Vazirani, and A. Wigderson. The quantum communication complexity of sampling. In *Proceedings of the 39th Annual IEEE Symposium on Foundations of Computer Science*, pages 342–351, 1998.
3. T. Cover and J. Thomas. *Elements of Information Theory.* Wiley Series in Telecommunications. John Wiley and Sons, 1991.
4. P. Duris, Z. Galil, and G. Schnitger. Lower bounds on communication complexity. *Information and Computation*, 73:1–22, 1987.
5. R. Jain, J. Radhakrishnan, and P. Sen. Privacy and interaction in quantum communication complexity and a theorem about the relative entropy of quantum states. In *Proceedings of the 43rd Annual IEEE Symposium on Foundations of Computer Science*, 2002. To appear.
6. H. Klauck. On quantum and probabilistic communication: Las Vegas and one-way protocols. In *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 644–651, 2000.
7. H. Klauck, A. Nayak, A. Ta-Shma, and D. Zuckerman. Interaction in quantum communication and the complexity of set disjointness. In *Proceedings of the 33rd Annual ACM Symposium on Theory of Computing*, pages 124–133, 2001.
8. P. B. Miltersen, N. Nisan, S. Safra, and A. Wigderson. On data structures and asymmetric communication complexity. *Journal of Computer and System Sciences*, 57(1):37–49, 1998.
9. M. Nielsen and I. Chuang. *Quantum Computation and Quantum Information.* Cambridge University Press, 2000.
10. N. Nisan and A. Wigderson. Rounds in communication complexity revisited. *SIAM Journal of Computing*, 22:211–219, 1993.
11. C. Papadimitriou and M. Sipser. Communication complexity. *Journal of Computer and System Sciences*, 28:260–269, 1984.
12. S. Ponzio, J. Radhakrishnan, and S. Venkatesh. The communication complexity of pointer chasing. *Journal of Computer and System Sciences*, 62(2):323–355, 2001.
13. A. Razborov. Quantum communication complexity of symmetric predicates. Manuscript at quant-ph/0204025, April 2002.
14. P. Sen and S. Venkatesh. Lower bounds in the quantum cell probe model. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, Lecture Notes in Computer Science, vol. 2076, pages 358–369. Springer-Verlag, 2001. Also quant-ph/0104100.
15. A. C-C. Yao. Quantum circuit complexity. In *Proceedings of the 34th Annual IEEE Symposium on Foundations of Computer Science*, pages 352–361, 1993.

The Decidability of the First-Order Theory of the Knuth-Bendix Order in the Case of Unary Signatures

Konstantin Korovin and Andrei Voronkov

University of Manchester
{korovin,voronkov}@cs.man.ac.uk

Abstract. We show that the first-order theory of any Knuth-Bendix order in the case of the signatures consisting of unary function symbols and constants is decidable. Our decision procedure uses interpretation of unary terms as trees and uses decidability of the weak monadic second-order theory of binary trees. One area of applications of our result is automated deduction, since using the first-order theory of the Knuth-Bendix orders we can decide an important class of ordering constraints.

1 Introduction

Introduction of ordering constraints has been one of the main breakthroughs in the saturation based theorem proving. Using solvability of ordering constraints we can dramatically reduce the number of redundant inferences in a resolution-based prover. As a consequence, the problem of solving ordering constraints for the known simplification orders is one of the important problems in the area. A simplification order is a total monotonic order on ground terms. Given such an order we can consider ordering constraints which are quantifier-free formulas in the language of the term algebra with equality and the order. Two kinds of orders are mainly used in automated deduction: the Knuth-Bendix orders [9] and various versions of the recursive path orders [5]. Because of its importance, the decision problem for ordering constraints has been well-studied. For the recursive path orders decidability and complexity issues were considered in [8,2,16,17,15,14]. For the Knuth-Bendix orders we have the following results: the decidability of constraints [10], a nondeterministic polynomial-time algorithm for constraint solving [11], a polynomial-time algorithm for solving constraints consisting of a single inequality [12].

In resolution-based theorem proving there are important simplifications which allow us to remove clauses form the search space (for example subsumption). It turns out that in order to express applicability conditions for these simplifications, we need to consider constraints which involve first-order quantifiers. Unfortunately the first-order theory of the recursive path orders is undecidable [20,4]. Only recently the decidability of the first-order theory of recursive path orders in the case of unary signatures has been proven [14]. A signature is called unary if it consists of unary function symbols and constants.

In this paper we prove the decidability of the first-order theory of the Knuth-Bendix orders in the case of unary signatures. Our decision procedure uses interpretation of unary terms as trees and uses decidability of the weak monadic second-order theory of binary trees.

2 Preliminaries

A *signature* is a finite set of function symbols with associated arities. *Constants* are function symbols of the arity 0. We assume that Σ contains at least one constant. We denote variables by x, y, z and terms by r, s, t. The set of all ground terms of the signature Σ can be considered as the *term algebra* of this signature, $\text{TA}(\Sigma)$, by defining the interpretation $g^{\text{TA}(\Sigma)}$ of any function symbol g by $g^{\text{TA}(\Sigma)}(t_1, \dots, t_n) = g(t_1, \dots, t_n)$. For details see e.g. [7] or [13]. It is easy to see that in term algebras any ground term is interpreted by itself. The Knuth-Bendix order is a family of orders parametrized by two parameters: a weight function and a precedence relation.

Definition 1 (weight function) We call a *weight function* on Σ any function $w : \Sigma \to \mathbb{N}$ such that (i) $w(a) > 0$ for every constant $a \in \Sigma$, (ii) there exist at most one unary function symbol $f \in \Sigma$ such that $w(f) = 0$. Given a weight function w, we call $w(g)$ the *weight* of g. The *weight* of any ground term t, denoted $|t|$, is defined as follows: for every constant c we have $|c| = w(c)$ and for every function symbol g of a positive arity we have $|g(t_1, \dots, t_n)| = w(g) + |t_1| + \dots + |t_n|$.

These conditions on the weight function ensure that the Knuth-Bendix order is a simplification order total on ground terms (see, e.g., [1]). In this paper, *f will always denote a unary function symbol of weight* 0.

Definition 2 (precedence relation) A *precedence relation* on Σ is any total order $\gg$ on Σ. A precedence relation $\gg$ is said to be *compatible* with a weight function w if, whenever f is a unary function symbol f of weight zero, f is the greatest element w.r.t. $\gg$.

In the sequel we assume a fixed weight function w on Σ and a fixed precedence relation $\gg$ on Σ, compatible with w.

Definition 3 The *Knuth-Bendix order* on $\text{TA}(\Sigma)$ is the binary relation $\succ$ defined as follows. For any ground terms $t = g(t_1, \dots, t_n)$ and $s = h(s_1, \dots, s_k)$ we have $t \succ s$ if one of the following conditions holds:

1. $|t| > |s|$;
2. $|t| = |s|$ and $g \gg h$;
3. $|t| = |s|$, $g = h$ and for some $1 \leq i \leq n$ we have $t_1 = s_1, \dots, t_{i-1} = s_{i-1}$ and $t_i \succ s_i$.

Note that the Knuth-Bendix order is a total monotonic well-founded order, see, e.g., [1]. Let $\text{TA}_{\succ}(\Sigma)$ denote the structure of the term algebra over Σ with the Knuth-Bendix order $\succ$.

In this paper we will only consider signatures consisting of unary function symbols and constants.

3 Interpretations

Interpretations play an important role in mathematical logic, allowing us to describe the properties of a given structure based on the properties of another structure.

We will use an interpretation of first-order structures with the Knuth-Bendix order, in the structure of two successors considered in the weak monadic second-order language. The weak monadic second-order language is a language closed under $\vee, \wedge, \neg$, which extends first-order language with variables $X, Y, \ldots$ ranging over finite sets, includes atomic formulas $t \in X$ where t is a first order term and allows quantifiers over the set variables.

Let us introduce a simple notion of interpretation which we will use later to show the decidability of the first-order theory of the unary Knuth-Bendix orders. For a more general theory of interpretations see, e.g., [7,6,18]. In the sequel we will use lower-case letters $x, y, z, \ldots$ to denote first-order variables and upper-case letters $X, Y, Z, \ldots$ to denote second-order variables.

DEFINITION 4 Let A be a structure in a first-order language L_A and B be a structure in a weak monadic second-order language L_B. We say that the structure A is interpretable in the structure B if there exist a positive integer m and the folling formulas:

1. $\phi_{domain}(\bar{X})$, where $\bar{X}$ is a tuple of second-order variables of the length m such that the set $A' = \{\bar{S} \mid B \models \phi_{domain}(\bar{S})\}$ is non-empty;
2. $\phi_g(\bar{X}_1, \ldots, \bar{X}_n, \bar{Y})$ for each function symbol g in the language L_A, where the arity of g is n and $\bar{X}_1, \ldots, \bar{X}_n, \bar{Y}$ are tuples of second-order variables of the length m, and this formula defines a function, denoted by g', on A', i.e., we have
$$g'(\bar{S}_1, \ldots, \bar{S}_n) = \bar{T} \Leftrightarrow B \models \phi_g(\bar{S}_1, \ldots, \bar{S}_n, \bar{T});$$
3. $\phi_P(\bar{X}_1, \ldots, \bar{X}_n)$ for each predicate symbol P in L_A, where the arity of P is n and $\bar{X}_1, \ldots, \bar{X}_n$ are tuples of second-order variables of the length m, and this formula defines a predicate on A', denoted by P', i.e., we have
$$P'(\bar{S}_1, \ldots, \bar{S}_n) \Leftrightarrow B \models \phi_P(\bar{S}_1, \ldots, \bar{S}_n);$$

such that the following condition holds.
The structure with the domain A', in which every function symbol f is interpreted by the function f' and every predicate symbol P is interpreted by P', is isomorphic to the structure A'.

We will use the following fundamental property of interpretability.

Proposition 1. *If a structure A is interpretable in the structure B and the theory of B (in the language L_B) is decidable, then the theory of A (in the language L_A) is also decidable.*

The proof can be found, e.g. in [7,6,18].

4 Interpretation of the Knuth-Bendix Order in WS2S

We will use interpretations to show the decidability of the first-order theory of the unary Knuth-Bendix orders. We show how to interpret Knuth-Bendix orders in the structure of two successors in the weak monadic language. Then, using the result [19] on the decidability of the weak monadic theory of two successors, we conclude that the first-order theory of the unary Knuth-Bendix orders is decidable.

Let us briefly recall the definition of the structure of two successors (see, e.g., [3] for details). The domain consists of finite binary strings including the empty string λ. There are two functions $0(x)$ and $1(x)$ which add 0 and 1 respectively to the end of the string. For example $0(101) = 1010$. Instead of $0(t)$ and $1(t)$ we will write, respectively, $t \cdot 0$ and $t \cdot 1$. The atomic formulas are equalities $t = s$ between first-order terms, and $t \in X$ where t is a first-order term. Formulas are built from atomic formulas using logical connectives $\wedge, \vee, \neg$, the first-order quantifiers $\exists x, \forall x$ and second-order quantifiers over finite sets $\exists X, \forall X$. We will use the following standard shorthands: $\exists x \in X \phi(x, X)$ for $\exists x(x \in X \wedge \phi(x, X))$ and $\forall x \in X \phi(x, X)$ for $\forall x(x \in X \supset \phi(x, X))$. Binary strings can be seen as positions in binary trees, and in the sequel we sometimes will use the word *position* instead of string.

Below we will use the following definable relations on sets with a straightforward meaning.

Emptiness:

$$X = \emptyset \leftrightarrow \forall x(x \notin X).$$

Intersection:

$$X \cap Y = Z \leftrightarrow \forall x(x \in Z \leftrightarrow (x \in X \wedge x \in Y)).$$

Union:

$$X \cup Y = Z \leftrightarrow \forall x(x \in Z \leftrightarrow (x \in X \vee x \in Y)).$$

Partition:

$$Partition(X, X_1, \ldots, X_n) \leftrightarrow X = \bigcup_{1 \le i \le n} X_i \wedge \bigwedge_{1 \le i < j \le n} X_i \cap X_j = \emptyset.$$

PrefixClosed:

$$PrefixClosed(X) \leftrightarrow \forall x((x \cdot 0 \in X \vee x \cdot 1 \in X) \supset x \in X).$$

Sets satisfying *PrefixClosed* will be called *trees*.

Prefix order $\sqsubseteq$:

$$x \sqsubseteq y \leftrightarrow \forall X((y \in X \wedge PrefixClosed(X)) \supset x \in X).$$

Likewise, we introduce

$$x \sqsubset y \leftrightarrow x \sqsubseteq y \wedge x \neq y.$$

Lexicographic order $\leq_{lex}$:

$$x \leq_{lex} y \leftrightarrow x \sqsubseteq y \bigvee \exists z(z \cdot 0 \sqsubseteq x \wedge z \cdot 1 \sqsubseteq y).$$

Likewise, we introduce

$$x <_{lex} y \leftrightarrow x \leq_{lex} y \wedge x \neq y.$$

Maximal prefix: Informally, $MaxPref(m, X)$ says that m is a maximal element in X w.r.t. the prefix order.

$$MaxPref(m, X) \leftrightarrow m \in X \wedge \forall z \in X \neg(m \sqsubset z).$$

Minimal prefix: Informally, $MinPref(m, X)$ says that m is a minimal element in X w.r.t. the prefix order.

$$MinPref(m, X) \leftrightarrow m \in X \wedge \forall z \in X \neg(z \sqsubset m).$$

Maximal lexicographically: Informally, $MaxLex(m, X)$ says that m is a maximal element in X w.r.t. the lexicographic order.

$$MaxLex(m, X) \leftrightarrow m \in X \wedge \forall z \in X \neg(m <_{lex} z).$$

Assuming a fixed Knuth-Bendix order we will show how to interpret it in the structure of two successors using the weak monadic second-order language.

Let us consider a signature $\Sigma = \{g_1, \dots, g_s\}$ consisting of unary function symbols and constants. From now on we assume that Σ is fixed and denote by s the number of function symbols and constants in it. We denote the set of constants in Σ by Σ_c and the set of unary function symbols by Σ_g. Let w be a weight function on Σ and $\gg$ be a precedence relation compatible with w. Also f will always denote the function symbol of weight zero. Denote the Knuth-Bendix order induced by this weight function and precedence relation by $\succ$. Now we show how to interpret $\mathrm{TA}_{\succ}(\Sigma)$ in the structure of two successors using the weak monadic language.

We define the interpretation in three steps. First we map terms into labelled trees and define functions and relations on them such that the obtained structure will be isomorphic to $\mathrm{TA}_{\succ}(\Sigma)$. Then we show how labelled trees can be represented as $s+1$-tuples of finite sets of binary strings. Finally we show how to define these representations, and corresponding functions and relations on them in the structure of two successor using weak monadic second–order logic.

Coding of Terms.

The labelled trees are binary trees labelled with the function symbols. We want tree representation of terms to satisfy the following properties

1. The functions of $\mathrm{TA}_{\succ}(\Sigma)$ can be defined in the monadic second-order language.
2. The function symbols of the term algebra are represented in such a way that we can compare weights of terms using the monadic second-order language.

3. For the terms of equal weight we should be able to compare their top function symbols and then lexicographically compare their subterms.

Let us start with an example. Consider a signature $\{f(), g(), h(), c\}$, and a weight function w such that $w(f) = 0, w(g) = 2, w(h) = w(c) = 1$. Figure 1 shows how to construct a labelled tree representing the term $f(h(f(f(g(c)))))$. The labelled tree is built by traversing the tree inside-out, for example, the root of the labelled tree is labelled with the constant c. We would like the rightmost branch of the tree to have the length equal to the weight of the term. To this end, we repeat every function symbol of a positive weight the number of times equal to its weight. Since the function symbol f has the weight 0, it is not included on the rightmost branch. To represent this symbol, we make branching to the left at the corresponding points of the tree.

Before giving a formal definition of the representation of terms as labelled trees, let us consider trees as sets of binary strings. Any binary tree without labels can be defined as a set of binary strings, namely the positions of the nodes in the tree. For example, the tree of Figure 1 contains the binary strings λ labelled with c, strings 1 and 11 labelled as g, string 111 labelled by h, and strings 110, 1100, and 1110 labelled by f.

Formally, for each term t we define a labelled binary tree Tree_t and two positions Right_t and Top_t in this tree. The definition is by induction on t.

1. If t is a constant c of a weight w, then Tree_t consists of the strings $\lambda, 1, \ldots, 1^{w-1}$, labelled by c, and $\mathit{Right}_t = \mathit{Top}_t = 1^{w-1}$.
2. If $t = f(t')$, then Tree_t is obtained from $\mathit{Tree}_{t'}$ by adding the string $\mathit{Top}_t \cdot 0$ labelled by f, and we have $\mathit{Top}_{t'} = \mathit{Top}_t \cdot 0$, $\mathit{Right}_{t'} = \mathit{Right}_t$.
3. If $t = g(t')$, where g has a positive weight w, then Tree_t is obtained from $\mathit{Tree}_{t'}$ by adding the strings $\mathit{Right}_t \cdot 1, \ldots, \mathit{Right}_t \cdot 1^w$ labelled by g, and we have $\mathit{Top}_{t'} = \mathit{Right}_{t'} = \mathit{Top}_t \cdot 1^w$.

The mapping $t \mapsto \mathit{Tree}_t$ defines the embedding of terms into labelled trees.

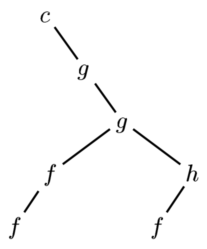


Fig. 1. The labelled tree representation of $fhffgc,\ w(f) = 0,\ w(g) = 2,\ w(h) = w(c) = 1$

Now it is easy to define the functions of the term algebra $\mathrm{TA}_{\succ}(\Sigma)$ on the labelled trees. We define the value of a function g on the labelled tree representation of a term t to be equal to the labelled tree representation of the term $g(t)$. Likewise, we can define the Knuth-Bendix order on such trees. It is evident that the obtained structure on the labelled trees is isomorphic to $\mathrm{TA}_{\succ}(\Sigma)$.

Now we will show how to represent labelled trees by $s+1$-tuples. Let T be a labelled tree whose set of positions is X. Then we represent T as the tuple $\langle X, X_{g_1}, \dots, X_{g_s}\rangle$, where each set X_{g_i} is the set of positions labelled by g_i and X is the set of all positions in this tree. If a term t is represented by a labelled tree T, and T is represented by a tuple $\langle X, X_{g_1}, \dots, X_{g_s}\rangle$, we will also say that the tuple $\langle X, X_{g_1}, \dots, X_{g_s}\rangle$ represents the term t.

To complete our construction, we have to show how to define in the second-order monadic language the set of tuples which represent the terms of $\mathrm{TA}_{\succ}(\Sigma)$, and then show that all functions and predicates of $\mathrm{TA}_{\succ}(\Sigma)$ are definable on the representation.

To this end we introduce some auxiliary definable predicates on sets of strings.

OneSucc: Informally, $\mathit{OneSucc}(X)$ says that the set of strings X consists of strings of 1's, contains the empty string, and is prefix closed.

$$\mathit{OneSucc}(X) \leftrightarrow \lambda \in X \wedge (\forall x \in X(x \neq \lambda \supset \exists y \in X\ x = y \cdot 1\ )).$$

Spine: The set of strings on rightmost branch of a tree will be called the *spine* of this tree. $\mathit{Spine}(X, Y)$ says that X is a tree and Y is its spine.

$$\begin{aligned}\mathit{Spine}(X, Y) \leftrightarrow\ & \mathit{PrefixClosed}(X) \wedge \mathit{OneSucc}(Y) \wedge Y \subseteq X \wedge \\ & \forall Y'((Y' \subseteq X \wedge \mathit{OneSucc}(Y')) \supset Y' \subseteq Y).\end{aligned}$$

Comb: Informally, $\mathit{Comb}(X)$ says that X is a tree and all right-branching positions in it are in its spine.

$$\begin{aligned}\mathit{Comb}(X) \leftrightarrow\ & \mathit{PrefixClosed}(X) \wedge \\ & \forall x(x \cdot 1 \in X \supset \exists Y\, \mathit{Spine}(X, Y) \wedge x \in Y).\end{aligned}$$

LabelledTree: Informally, $\mathit{LabelledTree}(X, X_{g_1}, \dots, X_{g_s})$ says that $\langle X, X_{g_1}, \dots, X_{g_s}\rangle$ is a tuple which is a labelled tree (not necessarily representing a term) appropriately labelled in the following sense: all positions along its spine are labelled with function symbols of positive weights and all other positions are labelled with the function symbol of the weight 0.

$$\begin{aligned}\mathit{LabelledTree}(X, X_{g_1}, \dots, X_{g_s}) \leftrightarrow\ & \mathit{Partition}(X, X_{g_1}, \dots, X_{g_s}) \\ & \wedge \mathit{Comb}(X) \\ & \wedge \mathit{Spine}(X, \cup_{g \in \Sigma \setminus \{f\}} X_g).\end{aligned}$$

The labelled trees defined by $\mathit{LabelledTree}(X, X_{g_1}, \dots, X_{g_s})$ are similar to those representing terms, except that in our representation of terms each occurrence of a function symbol of a positive weight should be repeated the number of times equal to the weight. Let us express this restriction in the weak monadic second-order logic.

A set consisting of strings of 1's will be called a 1-*set*. A 1-set which is a set of successive positions we be called an *interval*. The *length* of an interval is the number of elements in it. Consider a labelled tree $\langle X, X_{g_1} \dots, X_{g_s}\rangle$ and a function symbol $g \in \Sigma \setminus \{f\}$. First we introduce notions of g-interval and maximal g-interval. A g-*interval* is an interval which is contained in X_g and contains no branching positions with a possible exception of the maximal position of this interval.

***g*-interval:** Let $g \in \Sigma \setminus \{f\}$. Informally $\mathit{Interval}_g(I, \bar{X})$ says that $\bar{X}$ is a labelled tree and I is a g-interval.

$$\begin{aligned}\mathit{Interval}_g(I, \bar{X}) \leftrightarrow\ & \mathit{LabelledTree}(\bar{X}) \wedge I \subseteq X_g \wedge \\ & \exists m_0, m_1(\mathit{MinPref}(m_0, I) \wedge \mathit{MaxPref}(m_1, I) \\ & \qquad \wedge \forall y(m_0 \sqsubseteq y \sqsubseteq m_1 \supset y \in I)) \\ & \wedge \forall z \in I(\neg \mathit{MaxPref}(z, I) \supset z \cdot 0 \notin X).\end{aligned}$$

Maximal *g*-interval: is a g-interval that can not be properly extended.

$$\mathit{MaxInterval}_g(I, \bar{X}) \leftrightarrow \mathit{Interval}_g(I, \bar{X}) \wedge \forall J(\mathit{Interval}_g(J, \bar{X}) \supset I \not\subset J).$$

Our next goal is to express that the length of every maximal g-interval is a multiple of $w(g)$. To this end we introduce a notion of n-interval, for each positive n. We say that a position x is the n-successor of a position y if $x = y \cdot 1^n$. An *n-interval* is a 1-set which consists of a sequence of positions such that each next position is an n-successor of the previous. We always assume that an n-interval contains at least two elements. For example, the following set is a 2-interval $\{1, 111, 11111\}$. Let us show that for a given n, the property of being an n-interval is expressible in the monadic second-order logic.

1**-set:**

$$\mathit{OneSet}(X) \leftrightarrow \exists Y X \subseteq Y \wedge \mathit{OneSucc}(Y).$$

***n*-interval:**

$$\begin{aligned}\mathit{Interval}_n(X) \leftrightarrow\ & \mathit{OneSet}(X) \wedge \exists m(\mathit{MinPref}(m, X) \wedge 1^n(m) \in X) \\ & \wedge \forall y \in X(\mathit{MaxPref}(y, X) \vee (y \cdot 1^n \in X \wedge \textstyle\bigwedge_{1 \leq i < n} y \cdot 1^i \notin X)).\end{aligned}$$

Now, to say that the length of every maximal g-interval in a tree is a multiple of $w(g)$, it is enough to say that for every maximal g-interval in the tree, its minimal point and the successor of its maximal point are in some $w(g)$-interval.

Preterm: Informally, $\mathit{Preterm}(\bar{X})$ says that $\bar{X}$ is a labelled tree and the length of every maximal g-interval in this tree is a multiple of $w(g)$.

$$\begin{aligned}\mathit{Preterm}(\bar{X}) \leftrightarrow\ & \mathit{LabelledTree}(\bar{X}) \wedge \\ & \textstyle\bigwedge_{g \in \Sigma \setminus \{f\}} \forall I(\mathit{MaxInterval}_g(I, \bar{X}) \supset \\ & \qquad \exists m_0 \exists m_1(\mathit{MinPref}(m_0, I) \wedge \mathit{MaxPref}(m_1, I) \wedge \\ & \qquad\quad \exists Y\, \mathit{Interval}_{w(g)}(Y) \wedge m_0 \in Y \wedge m_1 \cdot 1 \in Y)).\end{aligned}$$

Finally, to define terms we need to say that the root position of a term is a constant and there are no other occurrences of constants.

Term:

$$\begin{aligned}\mathit{Term}(\bar{X}) \leftrightarrow\ & \mathit{Preterm}(\bar{X}) \wedge \lambda \in \textstyle\bigcup_{g \in \Sigma_c} \wedge \\ & \textstyle\bigwedge_{g \in \Sigma_c}(X_g \neq \emptyset \supset \lambda \in X_g \wedge \mathit{MaxPref}(1^{(w(g)-1)}(\lambda), X_g)).\end{aligned}$$

So, we have that $Term(\bar{X})$ defines the domain of our term algebra in the structure of two successors. Let us now show how to define the functions of the term algebra and the Knuth-Bendix order on this domain. Each constant can be easily defined as following.

Constants: For each constant $c \in \Sigma_c$ define

$$\phi_c(\bar{X}) \leftrightarrow Term(\bar{X}) \wedge X_c = \cup_{0 \le i < w(c)} \{1^i(\lambda)\} \wedge X = X_c.$$

Now we consider a function symbol $g \in \Sigma_g \setminus \{f\}$. In order to say that $\bar{Y} = g(\bar{X})$ we need to say that the spine of $\bar{Y}$ extends the spine of $\bar{X}$ with g repeated $w(g)$ times.

Function symbols of positive weight: For each function symbol $g \in \Sigma_g \setminus \{f\}$ define

$$\begin{aligned}\phi_g(\bar{X}, \bar{Y}) \leftrightarrow\ & Term(\bar{X}) \wedge Term(\bar{Y}) \wedge \textstyle\bigwedge_{h \in \Sigma \setminus \{g\}} X_h = Y_h \wedge \\ & \exists S \exists m (Spine(X, S) \wedge MaxLex(m, S) \wedge \\ & \quad Y_g = (X_g \cup \textstyle\bigcup_{1 \le i \le w(g)} \{1^i(m)\})).\end{aligned}$$

In order to say that $\bar{Y} = f(\bar{X})$ where f is the function symbol of zero weight we need to say that $\bar{Y}$ extends the greatest position in $\bar{X}$, w.r.t. lexicographic order, with f.

Function symbol of zero weight: For the function symbol of zero weight define

$$\begin{aligned}\phi_f(\bar{X}, \bar{Y}) \leftrightarrow\ & Term(\bar{X}) \wedge Term(\bar{Y}) \wedge \textstyle\bigwedge_{h \in \Sigma \setminus \{f\}} X_h = Y_h \wedge \\ & \exists m (MaxLex(m, X) \wedge Y_f = (X_f \cup \{m \cdot 0\})).\end{aligned}$$

Finally, we will define the Knuth-Bendix order. For this we need some auxiliary predicates.

Point of difference: Informally, $PointOfDifference(x, \bar{X}, \bar{Y})$ says that $\bar{X}, \bar{Y}$ represent terms and they differ at the position x.

$$\begin{aligned}PointOfDifference(x, \bar{X}, \bar{Y}) \leftrightarrow\ & Term(\bar{X}) \wedge Term(\bar{Y}) \wedge \\ & \textstyle\bigvee_{g \in \Sigma} ((x \in X_g \wedge x \notin Y_g) \vee \\ & \quad (x \in Y_g \wedge x \notin X_g)).\end{aligned}$$

Maximal point of difference: Informally, $MaxPointOfDifference(x, \bar{X}, \bar{Y})$ says that $\bar{X}, \bar{Y}$ are terms, and x is the greatest point of difference w.r.t. the lexicographic order.

$$\begin{aligned}MaxPointOfDifference(x, \bar{X}, \bar{Y}) \leftrightarrow\ & PointOfDifference(x, \bar{X}, \bar{Y}) \wedge \\ & \forall y (PointOfDifference(y, \bar{X}, \bar{Y}) \\ & \quad \supset y \le_{lex} x).\end{aligned}$$

Now we are ready to define the Knuth-Bendix order. Indeed, to say that $\bar{X} \succ \bar{Y}$ it is enough to say that $\bar{X}, \bar{Y}$ are terms, the maximal point of their difference is in X and the function symbol at this position in $\bar{X}$ is greater in the precedence relation $\gg$ than the function symbol at this position in $\bar{Y}$, if this position belongs to Y.

Knuth-Bendix order:

$$\begin{aligned}\bar{X} \succ \bar{Y} \leftrightarrow\ & \exists x (MaxPointOfDifference(x, \bar{X}, \bar{Y}) \wedge x \in X \wedge \\ & \textstyle\bigwedge_{g \in \Sigma} (x \in X_g \supset (x \notin Y \vee \bigvee_{h \ll g} x \in Y_h))).\end{aligned}$$

LEMMA 5 *The formulas $Term(\bar{X}), \bar{X} \succ \bar{Y}$ and $\phi_g(\bar{X}, \bar{Y})$ for $g \in \Sigma$, define an interpretation of the term algebra with the Knuth-Bendix order in the structure of two successors.*

PROOF. The claim follows from the definition of the Knuth-Bendix order. □

Using the decidability of the weak monadic second-order theory of two successors, this lemma and Proposition 1 we obtain the main result of this paper.

THEOREM 6 *The first-order theory of any Knuth-Bendix order in the case of the unary signatures is decidable.*

As an anonymous referee pointed out, our result can be easily extended to the decidability of term algebras with several Knuth-Bendix orders which have the same weight functions and different precedence relations. Indeed, in this case the interpretation of terms and term functions is the same as above and we only need to add formulas $\bar{X} \succ_i \bar{Y}$ for each Knuth-Bendix order $\succ_i$.

5 Open Problems

Let us mention some open problems. The first problem is to investigate the complexity of first-order theories of Knuth-Bendix orders in the case of unary signatures. Our algorithm uses decidability of the weak monadic second-order theory of two successors, which is known to be reasonably efficient in practice, but the complexity of this theory is non-elementary.

Another open problem is the decidability of the first-order theory of Knuth-Bendix orders in the case of arbitrary signatures.

Acknowledgment

The authors are grateful to the anonymous referees for helpful comments.

References

1. F. Baader and T. Nipkow. *Term Rewriting and and All That*. Cambridge University press, Cambridge, 1998.
2. H. Comon. Solving symbolic ordering constraints. *International Journal of Foundations of Computer Science*, 1(4):387–411, 1990.
3. 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.
4. H. Comon and R. Treinen. The first-order theory of lexicographic path orderings is undecidable. *Theoretical Computer Science*, 176(1-2):67–87, 1997.
5. N. Dershowitz. Orderings for term rewriting systems. *Theoretical Computer Science*, 17:279–301, 1982.
6. Yuri Ershov. *Problems of decidability and constructive models*. Nauka, 1980. (in Russian).

7. W. Hodges. *Model theory*. Cambridge University Press, 1993.
8. J.-P. Jouannaud and M. Okada. Satisfiability of systems of ordinal notations with the subterm property is decidable. In *Automata, Languages and Programming, 18th International Colloquium (ICALP)*, volume 510 of *Lecture Notes in Computer Science*, pages 455–468, 1991.
9. D. Knuth and P. Bendix. Simple word problems in universal algebras. In J. Leech, editor, *Computational Problems in Abstract Algebra*, pages 263–297. Pergamon Press, Oxford, 1970.
10. K. Korovin and A. Voronkov. A decision procedure for the existential theory of term algebras with the Knuth-Bendix ordering. In *Proc. 15th Annual IEEE Symp. on Logic in Computer Science*, pages 291–302, Santa Barbara, California, June 2000.
11. K. Korovin and A. Voronkov. Knuth-bendix ordering constraint solving is NP-complete. In F. Orejas, P.G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001*, volume 2076 of *Lecture Notes in Computer Science*, pages 979–992. Springer Verlag, 2001.
12. K. Korovin and A. Voronkov. Verifying orientability of rewrite rules using the Knuth-Bendix order. In A. Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001*, volume 2051 of *Lecture Notes in Computer Science*, pages 137–153. Springer Verlag, 2001.
13. M. Maher. Complete axiomatizations of the algebras of finite, rational and infi nite trees. In *Proc. IEEE Conference on Logic in Computer Science (LICS)*, pages 348–357, 1988.
14. Narendran and Rusinowitch. The theory of total unary rpo is decidable. In *First International Conference on Computational Logic*, volume 1861 of *Lecture Notes in Computer Science*, pages 660–672, 2000.
15. Narendran, Rusinowitch, and Verma. RPO constraint solving is in NP. In *CSL: 12th Workshop on Computer Science Logic*, volume 1584, pages 385–398. LNCS, Springer-Verlag, 1998.
16. R. Nieuwenhuis. Simple LPO constraint solving methods. *Information Processing Letters*, 47:65–69, 1993.
17. R. Nieuwenhuis and J.M. Rivero. Solved forms for path ordering constraints. In *In Proc. 10th International Conference on Rewriting Techniques and Applications (RTA)*, volume 1631 of *Lecture Notes in Computer Science*, pages 1–15, Trento, Italy, 1999.
18. Michael O. Rabin. Decidable theories. In J. Barwise, editor, *Handbook of Mathematical Logic*, volume 90 of *Studies in Logic and the Foundations of Mathematics*, chapter C.3, pages 595–629. North-Holland, Amsterdam, 1977.
19. James W. Thatcher and Jesse 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.
20. Ralf Treinen. A new method for undecidability proofs of first order theories. In *Foundations of Software Technology and Theoretical Computer Science,(FSTTCS), 10th conference*, volume 472 of *Lecture Notes in Computer Science*, pages 48–62, 1990. (journal version [21]).
21. Ralf Treinen. A new method for undecidablity proofs of first order theories. *Journal of Symbolic Computations*, 14(5):437–458, 1992.

Deciding the First Level of the μ-Calculus Alternation Hierarchy

Ralf Küsters and Thomas Wilke

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel, Germany
{kuesters,wilke}@ti.informatik.uni-kiel.de

Abstract. We show that the following problem is decidable and complete for deterministic exponential time. Given a formula of modal μ-calculus, determine if the formula is equivalent to a formula without greatest fixed point operators. In other words, we show that the first level of the μ-calculus fixed point alternation hierarchy is decidable in deterministic exponential time.

1 Introduction

Fixed point operators are the salient feature of modal μ-calculus. Therefore, it is only natural that one is interested in characterizing how much these operators contribute to the expressive power of the logic. One way to address this issue is to investigate natural hierarchies induced by fixed point operators. The most important hierarchy, which has been studied for decades, is the fixed point alternation hierarchy: a property expressible in modal μ-calculus is classified according to the number of alternations between greatest and least fixed points required to express the property. In 1996, this hierarchy was shown to be strict, independently by Bradfield [2] and Lenzi [8] (see also [1]). That is, the more alternations are allowed, the more one can express. In the proof of the result, a sequence of specific properties is exhibited where the ith member can be expressed with $i+1$ alternations, but not with i alternations. That is, for the specific properties of that proof we know exactly which level of the alternation hierarchy they belong to. In general, however, only very little is known about determining the number of alternations required to express an arbitrary property. In fact, Otto [10] showed in 1999 that it is decidable whether a given property can be expressed without fixed point operators, that is, he gave a decision procedure for level 0. Urbański [11] showed decidability for the second level under the assumption that the property is given as a *deterministic* tree automaton.

In this paper, we extend Otto's result to the first level of the hierarchy: we show that it is decidable whether a given property described by a μ-calculus formula can be expressed without greatest fixed points. (By duality, this also admits a decision procedure for determining whether a given property can be expressed without least fixed points.) More precisely, we show that the problem is complete for deterministic exponential time.

Our approach to the solution of the problem is an automata-theoretic one. We exploit the correspondence between modal μ-calculus and parity tree automata [9]: a property is expressible with least fixed points only if and only if it is recognized by an alternating tree automaton with a particular acceptance condition, namely, where a run is accepting if and only if it has no infinite path. The decidability criterion we propose is similar to decidability criteria for related problems. We show that a property, say given as a parity tree automaton, is expressible with least fixed points only if and only if the automaton is equivalent to a so-called test automaton. Decidability criteria of this type were already proposed by Landweber for checking whether a given regular ω-language belongs to the first two levels of the Borel hierarchy [7].

A straightforward implementation of our decidability criterion would result in a doubly exponential time decision procedure. The key to a singly exponential time procedure is a specific way of complementing our bottom-up test automaton and a detailed analysis of standard emptiness tests for alternating tree automata.

An overview of the paper can be found at the end of the following section.

Related Work. In independent, unpublished work [12], Igor Walukiewicz obtained essentially the same results as we do. He proved decidability in a restricted framework, where only properties of binary trees are studied, yet one can verify that his argument can be lifted to arbitrary transition systems.

Detailed proofs of all the results presented in this paper can be found in our technical report [6].

2 Basic Notions and Main Result

In this section we recall basic notions and state the main result.

2.1 Kripke Trees

We will interpret μ-calculus and MSOL-formulas (see below) over Kripke trees.

A *Kripke tree* t is a tuple (W, R, ρ, λ) where W is a nonempty set of *worlds,* $R \subseteq W \times W$ is an *accessibility relation* such that (W, R) is a tree rooted at $\rho \in W$, and $\lambda\colon W \to 2^{\mathcal{P}}$ is a *labeling function* that assigns to each world the set of propositional variables that hold true in it. Usually, we refer to the components of the tuple (W, R, ρ, λ) by W_t, R_t, etc. The set of all Kripke trees is denoted $\mathcal{T}$ and the set of Kripke trees labeled with subsets of P for some $P \subseteq \mathcal{P}$ is referred to by $\mathcal{T}_P$. Given a tree t and $w \in W_t$, the subtree of t rooted at w is denoted by $t \downarrow w$. We write $\mathrm{Suc}_t(w)$ for the set of *successors* of w in t, i.e., $\mathrm{Suc}_t(w) := \{w' \mid (w, w') \in R_t\}$. Note that this set might be infinite of any cardinality. A world w is a *leaf* if its set of successors is empty; otherwise w is called an *inner node.*

Two Kripke trees t and t' are called *bisimilar* ($t \cong t'$ for short) when there exists a relation $R \subseteq W_t \times W_{t'}$, called a *bisimulation relation*, such that $(\rho_t, \rho_{t'}) \in R$ and for every $(v, v') \in R$ and $p \in \mathcal{P}$:

- $p \in \lambda_t(v)$ iff $p \in \lambda_{t'}(v')$,
- whenever $(v, u) \in E_t$ for some u, then there exists u' such that $(v', u') \in E_{t'}$ and $(u, u') \in R$,
- whenever $(v', u') \in E_{t'}$ for some u', then there exists u such that $(v, u) \in E_t$ and $(u, u') \in R$.

A set S of Kripke trees is *closed under bisimulation* if $t \in S$ and $t' \cong t$ implies $t' \in S$ for every Kripke tree t and t'.

2.2 Modal μ-Calculus

Modal μ-calculus [5] is modal logic augmented by operators for least and greatest fixed points. For simplicity, this paper only deals with the unimodal case, but all definitions, results, and proofs easily extend to the multi-modal case.

We fix a countably infinite supply $\mathcal{P}$ of propositional variables. The formulas of modal μ-calculus are built from the constant symbols $\bot$ and $\top$, the symbols from $\mathcal{P}$ and their negations, using disjunction and conjunction, the modalities $\Box$ and $\Diamond$, and operators for least and greatest fixed points, μ and ν, where propositional variables bound by a fixed point operator occur only positive. The set of μ-calculus formulas is denoted by L_μ. The set of L_μ formulas that do not contain the greatest fixed point operator is denoted by Σ_1. Analogously, Π_1 shall denote the set of all L_μ formulas that do not contain the least fixed point operator (see, for example, [9] for the definition of the whole fixed point alternation hierarchy)

In general, L_μ formulas are interpreted in Kripke structures. Without loss of generality, we may only consider (arbitrary branching) Kripke trees as defined above. The interpretation of L_μ formulas on Kripke trees is defined as usual [5].

2.3 Monadic Second Order Logic (MSOL)

We fix a countably infinite set of unary predicate symbols $\mathcal{P}$, a countably infinite set $V_f = \{x, y, \ldots\}$ of first order variables, and a countably infinite set $V_s = \{X, Y, \ldots\}$ of second order variables.

The set of all MSOL formulas over $\mathcal{P}$, V_f, V_s, and the constant sr is defined inductively as follows: $p(x), s(x, y), X \subseteq Y, \mathsf{sr}$ are MSOL formulas. They are closed under Boolean operators and first- and second-order quantification. A *sentence* is a formula without free variables.

The semantics of MSOL is defined as follows, where V stands for a valuation, which assigns to every first-order variable an element of W_t and to every second-order variable a subset of W_t. The constant sr is interpreted as ρ_t. Further, $t, V \models p(x)$ iff $p \in \lambda_t(V(x))$; $t, V \models s(x, y)$ iff $(V(x), V(y)) \in E_t$; $t, V \models X \subseteq Y$ iff $V(X) \subseteq V(Y)$. The Boolean operators and the quantifiers are interpreted as usual [4].

2.4 Parity Tree Automata

Parity tree automata are finite-state devices designed to accept or reject Kripke trees.

A *parity tree automaton* $\mathbf{A}$ is a tuple (Q, q_I, δ, Ω), where Q is a finite set of *states,* $q_I \in Q$ is an *initial state,* δ is a *transition function* as specified below, and $\Omega\colon Q \to \omega$ is a *priority function,* which assigns a *priority* to each state. The transition function δ maps every state to a transition condition over $\mathcal{P}$ and Q. A *transition condition over* $\mathcal{P}$ *and* Q is defined as follows: $\bot$ and $\top$ are transition conditions; p and $\neg p$ are transition conditions, for every $p \in \mathcal{P}$; q, $\Box q$, and $\Diamond q$ are transition conditions, for every $q \in Q$; $q \wedge q'$ and $q \vee q'$ are transition conditions, for every $q, q' \in Q$. For $q \in Q$, we define $\mathbf{A}_q = (Q, q, \delta, \Delta)$ to be the parity automaton obtained from $\mathbf{A}$ by setting the initial state of $\mathbf{A}$ to q. We call a parity tree automaton which only uses priority 1 a *1-automaton.* Analogously, we define *0-automata*.

The computational behavior of parity tree automata is explained using the notion of a run. Assume $\mathbf{A}$ is a parity tree automaton and t a pointed Kripke structure. A *run* of $\mathbf{A}$ on t is a $(W_t \times Q)$-vertex-labeled tree $\mathbf{r} = (V_r, E_r, \rho_r, \lambda_r)$ such that ρ_r is labeled (ρ_t, q_I) and for every vertex v with label (w, q) the following conditions are satisfied:

- $\delta(q) \neq \bot$.
- If $\delta(q) = p$, then $p \in \lambda_t(w)$, and if $\delta(q) = \neg p$, then $p \notin \lambda_t(w)$.
- If $\delta(q) = q'$, then there exists $v' \in \mathrm{Suc}_r(v)$ such that $\lambda_r(v') = (w, q')$.
- If $\delta(q) = \Diamond q'$, then there exists $v' \in \mathrm{Suc}_r(v)$ such that $\lambda_r(v') = (w', q')$ for some $w' \in \mathrm{Suc}_t(w)$.
- If $\delta(q) = \Box q'$, then for every $w' \in \mathrm{Suc}_t(w)$ there exists $v' \in \mathrm{Suc}_r(v)$ with $\lambda_r(v') = (w', q')$.
- If $\delta(q) = q' \vee q''$, then there exists $v' \in \mathrm{Suc}_r(v)$ such that $\lambda(v') = (w, q')$ or $\lambda(v') = (w, q'')$.
- If $\delta(q) = q' \wedge q''$, then there exist $v', v'' \in \mathrm{Suc}_r(v)$ such that $\lambda(v') = (w, q')$ and $\lambda(v'') = (w, q'')$.

For a node v in $\mathbf{r}$ with $\lambda_r(v) = (w, q)$, define $\Omega_r(v) := \Omega_A(q)$. An infinite branch $\pi = v_0 v_1 v_2 \cdots$ of $\mathbf{r}$ is *accepting* if the maximum priority occurring infinitely often in the sequence $\Omega_r(v_0)\Omega_r(v_1)\Omega_r(v_2)\cdots$ is even. The run $\mathbf{r}$ is *accepting* if every infinite branch through $\mathbf{r}$ is accepting. A Kripke tree is *accepted* by $\mathbf{A}$ if there exists an accepting run of $\mathbf{A}$ on the Kripke tree. The set of all Kripke trees accepted by $\mathbf{A}$ is denoted by $T(\mathbf{A})$.

An L_μ formula ϕ is *equivalent* to a parity tree automaton $\mathbf{A}$ if every Kripke tree that holds in ϕ is accepted by $\mathbf{A}$ and vice versa.

The part of the one-to-one correspondence between the fixed point alternation hierarchy of the modal μ-calculus and the hierarchy of parity tree automata relevant for this paper reads as follows:

Fact 1 *1. For every L_μ formula one can construct in linear time an equivalent parity tree automaton. For Σ_1-formulas (Π_1-formulas) one obtains 1-automata (0-automata).*

2. Conversely, for every parity tree automaton, there exists an equivalent L_μ formula; 1- and 0-automata are equivalent to Σ_1- and Π_1-formulas, respectively.

2.5 Main Result

We consider the following decision problem:

Σ_1-Definability. Given an L_μ formula, decide whether there exists a Σ_1-formula equivalent to it.

Analogously, the problem Π_1-Definability is defined. The main purpose of this paper is to prove:

Theorem 1 Σ_1-Definability *is an EXPTIME-complete problem.*

Since ϕ belongs to Σ_1 iff $\neg\phi$ belongs to Π_1, from the theorem we immediately obtain:

Corollary 1 Π_1-Definability *is an EXPTIME-complete problem.*

Overview of the Proof of Theorem 1. The complexity lower bound claimed in Theorem 1 is obtained by reducing the problem L_μ-Tautology, which is known to be EXPTIME-complete, to Σ_1-Definability (see Section 7).

Most of this paper is devoted to the proof of the complexity upper bound. The idea is as follows.

Due to Fact 1, we can assume that in the problem Σ_1-Definability we are given a parity tree automaton $\mathbf{A}$ instead of an L_μ formula.

Given $\mathbf{A}$ over $P \subseteq \mathcal{P}$, we define a bottom-up tree automaton $\mathbf{B_A}$ accepting the so-called Σ_1-test language $\Sigma_1(\mathbf{A})$. A state of $\mathbf{B_A}$ is an equivalence class of a so-called characteristic equivalence relation of $\mathbf{A}$. The key to deciding Σ_1-Definability is a theorem stating that $T(\mathbf{A})$ is Σ_1-definable iff $T(\mathbf{A}) \subseteq \Sigma_1(\mathbf{A})$; the inclusion $\Sigma_1(\mathbf{A}) \subseteq T(\mathbf{A})$ holds in any case. This reduces Σ_1-Definability to the emptiness problem

$$T(\mathbf{A}) \cap (\mathcal{T}_P \setminus \Sigma_1(\mathbf{A})) \overset{?}{=} \emptyset \tag{1}$$

We show that the languages accepted by bottom-up automata are definable in monadic second order logic (MSOL). Consequently, the above emptiness problem, and thus, Σ_1-Definability is decidable. To obtain the claimed exponential upper bound, we prove that the complement of languages accepted by bottom-up tree automata are accepted by so-called top-down tree automata of the same size. A detailed analysis of standard emptiness tests for parity tree automata then shows that the above emptiness problem, where the complement of the Σ_1-test language is given as a top-down tree automaton, is decidable in exponential time.

In the following section, the characteristic equivalence relation is introduced. In Section 4, we define bottom-up tree automata and state important properties. The Σ_1-test language and the mentioned theorem can be found in Section 5. Section 6 studies the complementation of bottom-up tree automata and introduces top-down tree automata. Finally, in Section 7 everything is put together to prove the complexity upper bound, as well as the complexity lower bound.

3 The Characteristic Equivalence Relation

In this section, we introduce the characteristic equivalence relation of a parity tree automaton and study properties thereof.

If $\equiv$ is an equivalence relation on trees and t is a tree, then $[t]_\equiv := \{t' \mid t' \equiv t\}$ denotes the equivalence class of t w.r.t. $\equiv$. We say that $\equiv$ *saturates a tree set* T if T is a union of its classes.

Let $\mathbf{A}$ be a parity tree automaton. We define its *characteristic equivalence relation,* denoted $\equiv_A$, as follows. It considers trees t and t' equivalent if for every state q of $\mathbf{A}$: $t \in T(\mathbf{A}_q)$ iff $t' \in T(\mathbf{A}_q)$.

Obviously, the index of the characteristic equivalence relation $\equiv_\mathbf{A}$ can be bounded exponentially in the size of $\mathbf{A}$. Also, $\equiv_\mathbf{A}$ saturates $T(\mathbf{A})$.

There is a more complicated property that the characteristic equivalence relation of an automaton enjoys: the equivalence class of the tree itself is determined by the labeling of the root and the equivalence classes of the subtrees rooted at the children of the root. Equivalence relations with this property are called strong equivalence relations.

Formally, an equivalence relation $\equiv$ on trees is a *strong equivalence relation* if it satisfies the following condition. Trees t and t' are equivalent w.r.t. $\equiv$ if

$$\lambda_t(\rho_t) = \lambda_{t'}(\rho_{t'}), \text{ and}$$
$$\{[t \downarrow w]_\equiv \mid w \in \mathrm{Suc}_t(\rho_t)\} = \{[t' \downarrow w]_\equiv \mid w \in \mathrm{Suc}_{t'}(\rho_{t'})\}.$$

One can easily show:

Lemma 1 *The characteristic equivalence relation of any parity tree automaton is a strong equivalence relation. (This is even true for tree automata with infinite state spaces and infinitary transition conditions.)*

This lemma is the key to the definition of the transition function of our bottom-up tree automaton accepting the Σ_1-test language, for it tells us the following. For every parity tree automaton $\mathbf{A}$ over P there exists a unique function $f_\mathbf{A} \colon 2^{C_A} \times 2^P \to C_A$, where $C_\mathbf{A} := \{[t]_{\equiv_\mathbf{A}} \mid t \text{ is a tree }\}$ denotes the set of $\equiv_\mathbf{A}$-equivalence classes, such that the following holds: Given a tree t and the sets $P' = \lambda_t(\rho_t)$ and $C = \{[t \downarrow w]_{\equiv_\mathbf{A}} \mid w \in \mathrm{Suc}_t(\rho_t)\}$ we have

$$[t]_{\equiv_\mathbf{A}} = f_\mathbf{A}(C, P').$$

4 Bottom-up Tree Automata on Infinite Trees

Bottom-up tree automata are designed to accept or reject (infinite) trees. Given a tree, they first guess a frontier in the tree and assign to every world of this frontier a certain set of states, depending on the label of the world and on whether the world is a leaf or not. Then in a bottom-up powerset fashion the ancestors are labeled with sets of states according to the transition function. (All descendants of worlds of the frontier are labeled with the empty set.) A tree is accepted if the set of states the root is labeled with is non-empty and a subset of the set of final states of the automaton.

Formally, a *bottom-up tree automaton* $\mathbf{B}$ is a tuple (Q, P, F, δ), where Q is a finite set of states, P is a finite set of propositional variables, $F \subseteq Q$ is the set of final states, and $\delta\colon 2^Q \times 2^P \to Q$ is the transition function.

To define a run of $\mathbf{B}$ on a tree $t \in \mathcal{T}_P$, we need two additional functions, $\gamma\colon 2^P \to Q$ and $\Gamma\colon 2^P \to 2^Q$:

$$\begin{aligned}\gamma(P') &= \delta(\emptyset, P'),\\ \Gamma(P') &= \{\delta(Q', P') \mid Q' \subseteq Q \wedge Q' \neq \emptyset\}.\end{aligned}$$

The former function is used to determine the set of states assigned to leaves belonging to the frontier and the latter function determines the set of states assigned to inner nodes of the frontier. In other words, inner nodes are labeled with the set of all states that this node can possibly take according to the transition function δ.

We also define the function

$$\Delta\colon 2^{2^Q} \times 2^P \to 2^Q,$$

which is the "powerset variant" of δ: A state $q \in Q$ belongs to $\Delta(\mathcal{Q}, P')$ if there exists a set E and a selection function $s\colon \mathcal{Q} \to Q$ with $s(Q') \in Q'$, for every $Q' \in \mathcal{Q}$, $E = \{s(Q') \mid Q' \in \mathcal{Q}\}$, and $q = \delta(E, P')$.

A *frontier* $\mathcal{F}$ in t is a subset of W_t such that i) there does not exist a path between two different worlds in $\mathcal{F}$ ($\mathcal{F}$ is an anti-chain), and ii) every maximum path in t starting from ρ_t contains a world in $\mathcal{F}$. If $\mathcal{F}$ is a frontier, the set $\hat{\mathcal{F}}$ shall denote the set of ancestors of worlds in $\mathcal{F}$, including the worlds in $\mathcal{F}$.

Now, a *run* on t is a function $\beta\colon W_t \to 2^Q$ which has the following properties: There exists a frontier $\mathcal{F}$ in t such that

1. when $w \in \mathcal{F}$ is a leaf, then $\beta(w) = \{\gamma(\lambda_t(w))\}$,
2. when $w \in \mathcal{F}$ is an inner node, then $\beta(w) = \Gamma(\lambda_t(w))$,
3. when $w \in \hat{\mathcal{F}} \setminus \mathcal{F}$, then $\beta(w) = \Delta(\{\beta(w') \mid w' \in \mathrm{Suc}_t(w)\}, \lambda_t(w))$,
4. when $w \notin \hat{\mathcal{F}}$, then $\beta(w) = \emptyset$.

A run β on t is called *accepting* if $\beta(\rho_t) \neq \emptyset$ and $\beta(\rho_t) \subseteq F$. The set of trees accepted by $\mathbf{B}$ is denoted $T(\mathbf{B}) := \{t \in \mathcal{T}_P \mid$ there exists an accepting run of $\mathbf{B}$ on $t\}$.

In general, the languages accepted by bottom-up tree automata are not closed under bisimulation, and thus, are not definable in L_μ; this is even true for bottom-up tree automata induced by parity tree automata (see [6] for an example). Nevertheless, we can show the following.

Proposition 1 *1. The languages accepted by bottom-up tree automata are definable in MSOL.*
2. Whenever the language accepted by a bottom-up tree automaton is closed under bisimulation, then it is Σ_1-definable.

Proving the MSOL-definability is straightforward. To show 2., we specify a 1-automaton and prove that it is equivalent to the given bottom-up tree automaton provided that the language accepted by the bottom-up automaton is bisimulation closed. The difficult part is to show that the language accepted by the 1-automaton is contained in the one accepted by the bottom-up automaton: We start with an accepting run of the 1-automaton on t, and construct an accepting run of the bottom-up automaton on a tree t' which is bisimilar to t. Now, using the closure property, we conclude that t is accepted by the bottom-up automaton.

5 The Decidability Criterion

Given a parity tree automaton $\mathbf{A}$, we define a bottom-up tree automaton $\mathbf{B_A} = (C_\mathbf{A}, P, \{[t]_{\equiv_\mathbf{A}} \mid t \in T(\mathbf{A})\}, f_\mathbf{A})$, where $f_\mathbf{A}$ is specified as at the end of Section 3. We call the language accepted by $\mathbf{B_A}$ the *Σ_1-test language* of $\mathbf{A}$ and denote it by $\Sigma_1(\mathbf{A})$. The following theorem provides the criterion for deciding Σ_1-DEFINABILITY.

Theorem 2 *Let* $\mathbf{A}$ *be a parity tree automaton. Then,*

$$T(\mathbf{A}) \textit{ is } \Sigma_1\textit{-definable} \quad \textit{iff} \quad T(\mathbf{A}) = \Sigma_1(\mathbf{A}).$$

The implication from right to left follows from Proposition 1: If $T(\mathbf{A}) = \Sigma_1(\mathbf{A})$, then $\Sigma_1(\mathbf{A})$ is bisimulation closed since every language accepted by a parity tree automaton has this property. Proposition 1 implies that $\Sigma_1(\mathbf{A})$ is Σ_1-definable, and hence, so is $T(\mathbf{A})$.

For the implication in the other direction we first show:

Lemma 2 *For every parity tree automaton* $\mathbf{A}$*,*

$$\Sigma_1(\mathbf{A}) \subseteq T(\mathbf{A}).$$

This lemma is proved by transfinite induction. To this end, we characterize the run β of $\mathbf{B_A}$ as the fixed point of a certain mapping which mimics the bottom-up computation of $\mathbf{B_A}$ on a tree t starting from a frontier $\mathcal{F}$ in t. Then, we show that for every $w \in \hat{\mathcal{F}}$, $[t \downarrow w]_{\equiv_\mathbf{A}} \in \beta(w)$. Given that β is an accepting run, this implies $[t]_{\equiv_\mathbf{A}} \in \{[t']_{\equiv_\mathbf{A}} \mid t' \in T(\mathbf{A})\}$, and thus, $t \in T(\mathbf{A})$.

It remains to show:

Lemma 3 *If* $T(\mathbf{A})$ *is* Σ_1*-definable, then* $T(\mathbf{A}) \subseteq \Sigma_1(\mathbf{A})$.

The idea of the proof is as follows. Given that $T(\mathbf{A})$ is Σ_1-definable, there exists a 1-automaton $\mathbf{A}'$ accepting $T(\mathbf{A})$. For every accepting run r of $\mathbf{A}'$ on some tree t, we know that every path in r is finite. In other words, there exists a frontier $\mathcal{F}$ in t such that all descendants of worlds in $\mathcal{F}$ are not occupied by states of $\mathbf{A}'$. Now, we use $\mathcal{F}$ to construct an accepting run of $\mathbf{B_A}$ on t exploiting that "below" $\mathcal{F}$ we can change the structure of the tree anyhow without leaving $T(\mathbf{A})$.

6 Complementing Bottom-up Tree Automata

We show that the complement of a language accepted by a bottom-up tree automaton is accepted by a top-down tree automaton of the same size, which will be used to decide the emptiness problem (1). We also note that from the correspondence between the complement of bottom-up tree automata and top-down tree automata, one can easily derive a decidability criterion for Π_1-DEFINABILITY, similar to the one for Σ_1-DEFINABILITY.

First, we define top-down tree automata. A *top-down tree automaton* $\mathbf{D}$ is a tuple (Q, P, I, δ), defined just as for bottom-up tree automata, except that $I \subseteq Q$ is the set of *initial* states. A *run* r of $\mathbf{D}$ on $t \in \mathcal{T}_P$ is a mapping from W_t into Q such that $r(w) = \delta(\{r(w') \mid w' \in \mathrm{Suc}_t(w')\}, \lambda_t(w))$ for every $w \in W_t$. The run is called *accepting* if $r(\rho_t) \in I$. A tree $t \in \mathcal{T}_P$ is accepted by $\mathbf{D}$, if there exists an accepting run of $\mathbf{D}$ on t. The set of trees accepted by $\mathbf{D}$ is denoted $T(\mathbf{D})$.

As in the case of bottom-up tree automata, the languages accepted by top-down tree automata (even those induced by parity tree automata) are not bisimulation closed; see [6] for an example. However, we show:

Proposition 2 *1. The language accepted by a top-down tree automaton is definable in MSOL.*
2. Whenever the language accepted by a top-down tree automaton is closed under bisimulation, then it is Π_1-definable.

The main statement of this section is:

Proposition 3 *For every bottom-up tree automaton $\mathbf{B}$ there exists a top-down tree automaton which accepts the complement $\mathcal{T}_P \setminus T(\mathbf{B})$ of $T(\mathbf{B})$ and is of size linear in the size of $\mathbf{B}$.*

The proof works as follows. Given a bottom-up tree automaton $\mathbf{B} = (Q, P, F, \delta)$, we show that the top-down automaton

$$\mathbf{D_B} = (Q, P, Q \setminus F, \delta)$$

accepts $\mathcal{T}_P \setminus T(\mathbf{B})$. This is done in two steps. First, one shows the statement for finitely branching trees. Then, one assumes that the symmetric difference of $\mathcal{T}_P \setminus T(\mathbf{B})$ and $T(\mathbf{D_B})$ contains an infinite branching tree, and shows this implies a contradiction: Since both sets are MSOL-definable, their symmetric difference is MSOL-definable, and thus, it must contain a finitely branching tree due to the finite model property of MSOL. However, this contradicts the statement shown in the first step.— In the first step, finitely branching trees are considered to iteratively construct an accepting run of $\mathbf{D_B}$ on t given that $t \in \mathcal{T}_P \setminus T(\mathbf{B})$.

As a corollary of Theorem 2, Lemma 2, and (the proof of) Proposition 3, we obtain:

Corollary 2 *For every parity tree automaton $\mathbf{A}$,*

$$T(\mathbf{A}) \text{ is } \Sigma_1\text{-definable} \quad \text{iff} \quad T(\mathbf{A}) \cap T(\mathbf{D_{B_A}}) = \emptyset.$$

Also, from (the proof of) Proposition 3 and Theorem 2 we can derive a characterization of Π_1-definability. Given a parity tree automaton $\mathbf{A}$ over the propositional variables P, define the top-down tree automaton $\mathbf{D_A} := (C_\mathbf{A}, P, \{[t]_{\equiv_\mathbf{A}} \mid t \in T(\mathbf{A})\}, f_\mathbf{A})$. We call the set $\Pi_1(\mathbf{A}) := T(\mathbf{D_A})$ the *Π_1-test language of* $\mathbf{A}$. Now, one can show:

Corollary 3 *For every parity tree automaton* $\mathbf{A}$,

$$T(\mathbf{A}) \text{ is } \Pi_1\text{-definable} \quad \text{iff} \quad T(\mathbf{A}) = \Pi_1(\mathbf{A}).$$

7 Complexity

The aim of this section is to show the complexity upper and lower bound claimed in Theorem 1. We first show the upper bound and then present the proof of the lower bound.

7.1 Upper Bound

Due to Fact 1 we can assume that instead of a μ-calculus formula we have given a parity tree automaton $\mathbf{A}$. Also, Corollary 2 tells us that it suffices to show that the emptiness problem $T(\mathbf{A}) \cap T(\mathbf{D}) \overset{?}{=} \emptyset$ is decidable in deterministic exponential time, where $\mathbf{D} := \mathbf{D}_{\mathbf{B_A}}$.

The proof of this proceeds in two steps. First, we show a “small branching property” for $T(\mathbf{A}) \cap T(\mathbf{D})$. And then, we describe how a typical emptiness test for parity tree automata can be modified so as to work for our problem.

Small Branching Property. We define an ordering on trees. When t and t' are trees, then $t \sqsubseteq t'$ if t is a subgraph (in the usual sense) of t' and has the same root as t', that is, t results from t' by removing subtrees. Given t and t' such that $t \sqsubseteq t'$, we use $[t, t']$ to denote $\{t^* \mid t \sqsubseteq t^* \sqsubseteq t'\}$.

We say that a tree language T has the *n-branching property* if for every tree $t \in T$, there exists a tree $t_0 \sqsubseteq t$ of branching degree $\leq n$ such that $[t_0, t] \subseteq T$.

A typical emptiness test for parity tree automata starts with a statement similar to the following, see, for instance, [3].

Lemma 4 *Every tree language recognized by a parity tree automaton with n states has the n-branching property.*

We prove a similar statement for top-down automaton induced by parity tree automata.

Lemma 5 *For every parity tree automaton* $\mathbf{A}$ *with n states, the language accepted by* $\mathbf{D}$*, as defined above, has the $2n$-branching property.*

To prove this lemma, we introduce so-called modal top-down automata (see [6]) and show that $\mathbf{D}$ is equivalent to a modal top-down automaton with $2n$ states, given that $\mathbf{A}$ has n states. We then show that modal top-down automata with n' states have the n'-branching property.

As a consequence, we can note the following.

Corollary 4 *For every parity tree automaton* $\mathbf{A}$ *with* n *states, the set* $T(\mathbf{A}) \cap T(\mathbf{D})$ *has the* $(n+2n)$*-branching property.*

For the proof of this, just consider the union of the two trees that are guaranteed to exist by the individual small branching properties.

Emptiness Test. Consider a typical emptiness test for a parity tree automaton $\mathbf{A}$ with n states, for instance, the one described in [3]. In the first step, one exploits the n-branching property. Using Safra's construction, one constructs a non-deterministic Rabin tree automaton $\mathbf{A}'$ with $2^{O(n \log n)}$ states and $O(n)$ pairs which is equivalent to $\mathbf{A}$ on all $\leq n$ branching trees. This can be an automaton with transitions depending on the branching degree. In the second step, one solves the emptiness test for this automaton, for instance, by reducing it to the problem of finding the winner in an appropriate two-player infinite game.

In order to check whether or not $T(\mathbf{A}) \cap T(\mathbf{D}) = \emptyset$, as above, one can proceed in a similar fashion. By Corollary 4 we know that $T(\mathbf{A}) \cap T(\mathbf{D})$ has the $3n$-branching property. Therefore, in a first step, we construct a non-deterministic Rabin tree automaton $\mathbf{A}'$ as above which is equivalent to $\mathbf{A}$ on all $\leq 3n$ branching trees. Second, we convert $\mathbf{D}$ to a non-deterministic tree automaton $\mathbf{D}'$ of exponential size with trivial acceptance condition (0-acceptance) which is equivalent to $\mathbf{D}$ on all $\leq 3n$ branching trees. (Observe that this is possible since we can check in deterministic exponential time whether or not $f_{\mathbf{A}}(\mathcal{C}, P') = C$ for all $\mathcal{C} \subseteq C_{\mathbf{A}}$, $|\mathcal{C}| \leq 3n$, and $P' \subseteq P$.) Third, we form, in a straightforward manner, a product of $\mathbf{A}'$ and $\mathbf{D}'$ that recognizes exactly all $\leq 3n$ branching trees in $T(\mathbf{A}) \cap T(\mathbf{D})$. Finally, we perform an emptiness test for this product automaton, for instance, by reducing it to a winner problem for an appropriate game.

7.2 Lower Bound

The proof of the lower bound is quite simple. We reduce the tautology problem for modal μ-calculus, L_μ-Tautology, to Σ_1-Definability. This proves hardness for deterministic exponential time, because the satisfiability problem for modal μ-calculus, L_μ-Satisfiability, is complete for deterministic exponential time [3,5], and deterministic exponential time is closed under complementation.

Proposition 4 L_μ-Tautology *is polynomial-time reducible to* Σ_1-Definability.

The idea of the proof is as follows. Let p be some new propositional variable. Define π_1 by $\pi_1 = \Diamond \nu X(p \wedge \Box X)$. Clearly, the property defined by π_1 is Π_1- but not Σ_1-definable, see [1].

Now, let ϕ be an arbitrary L_μ-formula. Consider the formula ϕ^* defined by $\phi^* = \Diamond(\neg p \wedge \neg\phi) \wedge \pi_1$. Then, we show that ϕ is a tautology iff ϕ^* is equivalent to a Σ_1-formula.

References

1. André Arnold. The μ-calculus alternation-depth hierarchy is strict on binary trees. *Theoretical Informatics and Applications*, 33:329–339, 1999.
2. J. C. Bradfield. The Modal mu-calculus Alternation Hierarchy is Strict. In Ugo Montanari and Vladimiro Sassone, editors, *CONCUR '96: Concurrency Theory. 7th International Conference*, volume 1119 of *LNCS*, pages 232–246. Springer-Verlag, August 1996.
3. E.A. Emerson and C.S. Jutla. The Complexity of Tree Automata and Logics of Programs (Exteded Abstract). In *IEEE Symposium on Foundations of Computer Science* (FoCS'88), pages 328–337, Los Alamitos, California, October 1988. IEEE Computer Society Press.
4. David Janin and Igor Walukiewicz. On the Expressive Completeness of the Propositional mu-Calculus with Respect to Monadic Second Order Logic. In U. Montanari and V. Sassone, editors, *CONCUR'96: Concurrency Theory*, volume 1119 of *LNCS*, pages 263–277. Springer-Verlag, August 1996.
5. Dexter Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science*, 27:333–354, 1983.
6. R. Küsters and T. Wilke. Deciding the First Level of the μ-calculus Alternation Hierarchy. Technical Report 0209, Institut für Informatik und Praktische Mathematik, CAU Kiel, Germany, 2002. Available from http://www.informatik.uni-kiel.de/reports/2002/0209.html.
7. L.H. Landweber. Decision problems for ω-automata. *Math. Systems Theory*, 3:376–384, 1969.
8. Giacomo Lenzi. A Hierarchy Theorem for the μ-Calculus. In F. Meyer auf der Heide and B. Monien, editors, *Automata, Languages and Programming. 23rd International Colloquium. ICALP '96*, volume 1099 of *LNCS*, pages 87–97. Springer-Verlag, July 1996.
9. Damian Niwiński. On Fixed-Point Clones (Extended Abstract). In Laurent Kott, editor, *Automata, Languages and Programming. 13th International Colloquium*, volume 226 of *LNCS*, pages 464–473. Springer-Verlag, July 1986.
10. M. Otto. Eliminating Recursion in the mu-Calculus. In *16th Annual Symposium on Theoretical Aspects of Computer Science (STACS'99)*, volume 1563 of *Lecture Notes in Computer Science*, pages 531–540. Springer-Verlag, 1999.
11. T. F. Urbański. On Deciding if Deterministic Rabin Language Is in Büchi Class. In U. Montanari, J.D.P. Rolim, and E. Welzl, editors, *Automata, Languages and Programming, 27th International Colloquium (ICALP 2000)*, volume 1853 of *Lecture Notes in Computer Science*, pages 663–674. Springer-Verlag, 2000.
12. I. Walukiewicz, 2002. Personal communication.

Dynamic Message Sequence Charts

Martin Leucker[1,2,⋆], P. Madhusudan[2,⋆⋆], and Supratik Mukhopadhyay[2]

[1] Dept. of Computer Systems, Uppsala University, Sweden
[2] Dept. of Computer and Information Science, University of Pennsylvania, USA
{leucker,madhusudan,supratik}@cis.upenn.edu

Abstract. We introduce a formalism to specify classes of MSCs over an unbounded number of processes. The formalism can describe many interesting behaviours of dynamically changing networks of processes. Moreover, it strictly includes the formalism of Message Sequence Graphs studied in the literature to describe MSCs over a fixed finite set of processes. Our main result is that model-checking of MSCs described in this formalism against a suitable monadic-second order logic is decidable.

1 Introduction

In the early stages of design of a communication system, an emerging practice is to specify the system by describing the possible scenarios. A popular notation to describe such scenarios is that of message sequence charts (MSCs). This ITU standardized notation ([8]) describes a snap-shot of messages sent and received by various components of a distributed system in graphical notation.

There is now a growing interest in analyzing MSCs and applying the formal analysis techniques of model-checking to them. The motivation is that a designer could specify the scenarios using a collection of MSCs and verify them against certain requirements. Detection of errors at such an early stage in design can have considerable pay-offs.

In [1], the authors studied model-checking of message-sequence graphs (MSGs) against linear-time specifications. MSGs are a compact way of specifying an infinite collection of MSCs as a regular combination of atomic MSCs using concatenation, choice and repetition. Since an MSC describes a partially-ordered behaviour, a natural question to ask is whether all the linearizations of all the MSCs in the collection satisfy the linear specification. As shown in [1], this problem is undecidable unless the MSGs are heavily restricted.

In [5], the model-checking problem for MSCs was studied for structural specifications expressed in monadic second-order logic (MSO). MSO specifications describe properties of the partial-order defined by the MSC rather than of their linearizations. It was shown that the model-checking problem is decidable for MSGs, without any restriction on them. The idea that structural specifications can make model-checking effective has been extended for a larger class of MSC languages than MSGs [6].

⋆ supported by DARPA MOBIES F33615-00-C-1707, NSF CCR-9988409, NSF CISE-9703220, and the European Research Training Network on "Games"
⋆⋆ supported by NSF awards CCR99-70925 and ITR/SY 0121431

All the classes of MSC languages considered have, however, a serious limitation—they describe behaviours of only a *fixed* finite set of processes. While this is sufficient for many protocols, there are a number of domains where the number of processes cannot be considered fixed. Prime examples of such protocols are those that arise in the areas of mobile computing and ad-hoc networks. For example, a mobile phone may interact with an arbitrary number of transmitters and the description of switching from one to the other is naturally modelled only with a varying number of transmitters.

In this paper, we propose a formalism to describe MSC languages where the underlying set of processes need not be bounded. The formalism is given by a grammar which allows the user to concatenate MSCs (as in an MSG) and also gives the ability to *split* processes into two teams. Both teams can then independently interact with new processes that are spawned, and then join again, upon which they can continue interacting with each other. In order to contain ourselves in a decidable fragment, we constrain that at any point, the number of interesting processes for which the user is specifying a behaviour is fixed. Our main result is that monadic-second order specifications can be model-checked effectively for such a set of dynamic MSCs specified in this grammar.

Our formalism is quite natural and can describe many interesting behaviours involving unboundedly many processes. Like process calculus [7], it uses recursion to describe scenarios over an arbitrary number of processes. Also, our formalism is a proper extension of that of MSGs, which can be formulated in our grammar easily.

The main technical arguments for the decidability of model-checking stem from the rich theory of decidable classes of graphs against MSO formulas studied by Courcelle and others (see [3,2]). The classes of graphs we generate are similar in spirit to that of series-parallel graphs, for which similar decidability results are known. We see the main contribution of this paper as that of identifying a meaningful and natural class of MSC languages that allow depiction of scenarios of unboundedly many processes, and applying existing techniques in order to verify them against powerful structural specifications.

2 Fork-and-Join MSCs and Monadic Second-Order Logic

Let P, P', ... denote finite sets of *process names* (or just *processes* in short). We let $p, q, p_1, p', q', p'_1, \ldots$ range over processes. A *message alphabet* is a finite set Γ_M. Its elements are usually denoted by $a, a_1, \ldots$. For notational convenience, we fix Γ_M for the rest of the paper. For a set of processes P, let $Ac(P) := \{(p!q, a), (p?q, a) \mid p, q \in P, a \in \Gamma_M\}$ denote the set of *actions* over P. An action $(p!q, a)$ should be read as "p sends a message a to process q", and $(q?p, a)$ represents the corresponding receive action, which is then executed by process q. We denote by $Ac_p(P)$ the set of actions that $p \in P$ *participates* in, defined as $Ac_p(P) = \{(p!q, a), (p?q, a) \mid q \in P, a \in \Gamma_M\}$.

A message-sequence chart (MSC) is a partially ordered set of send and receive events, with a matching function that identifies the send events with corresponding receive events:

Definition 1. *Let P be a set of processes. A* Message Sequence Chart (MSC) *over P is a tuple $m = (P, E, \{\leq_p\}_{p \in P}, \lambda, \mu)$ where*

– *E is a finite set of* events*;*
– *$\lambda : E \to Ac(P)$ is a* labelling function *that identifies for each event an action. Let $E_p = \{e \in E \mid \lambda(e) \in Ac_p(P)\}$ denote the set of events which p participates in. Also, let $E_{\mathcal{S}} = \{e \in E \mid \lambda(e) = (p!q, a)$ for some $p, q \in P, a \in \Gamma_M\}$ and $E_{\mathcal{R}} = \{e \in E \mid \lambda(e) = (p?q, a)$ for some $p, q \in P, a \in \Gamma_M\}$ denote the set of* send *and* receive events *of E, respectively;*
– *$\mu : E_{\mathcal{S}} \to E_{\mathcal{R}}$ is a* matching function *that associates with each send event its corresponding receive event. Therefore, we require μ to be a bijection and for $e, e' \in E$, if $\mu(e) = e'$ then $\lambda(e) = (p!q, a)$ and $\lambda(e') = (q?p, a)$ for some $p, q \in P, a \in \Gamma_M$;*
– *$\leq_p$ is a total order on E_p for each $p \in P$. Let $\widehat{\leq} = (\bigcup_{p \in P} \leq_p) \cup \{(e, e') \mid e, e' \in E$ and $\mu(e) = e'\}$. Let $\leq = (\widehat{\leq})^*$ be the reflexive and transitive closure of $\widehat{\leq}$. Then $\leq$ denotes the* causal ordering *of E in the MSC and we require it to be a partial order on E.*

Note that in the above definition, we do not require E_p to be nonempty, for any $p \in P$.

An MSC m is considered equivalent to an MSC m' if it differs from m' only on the process and event sets—i.e. if the process and event sets of m can be relabelled (with appropriate relabelling of the actions) to yield m'.

We want to identify certain processes of MSCs when composing them. Let $\Pi_k = \{\pi_1, \ldots, \pi_k\}$ be a set of k *process identifiers*. To simplify the presentation, we fix Π_k, for some k, for the rest of the paper.

Definition 2. *A* named *MSC over P is a tuple (m, β) where m is an MSC over P and $\beta : \Pi_k \to P$ is an injective mapping, which assigns to every process identifier a process.*

Note that P must comprise at least k processes. We usually denote a named MSC by M or M'. Figure 1 illustrates two named MSCs M_1 and M_2. We are now ready to define the sequential composition for named MSCs.

When two named MSCs are concatenated, the processes corresponding to the same process identifier get identified and their events get causally related; the other processes are simply added as separate processes.

Definition 3. *Let $M = ((P, E, \{\leq_p\}_{p \in P}, \lambda, \mu), \beta)$ and $M' = ((P', E', \{\leq'_p\}_{p \in P'}, \lambda', \mu'), \beta')$ be two named MSCs. Since processes and events can be renamed, we may assume that for all $p \in P, p' \in P'$,*

– *$p = \beta(\pi)$, $p' = \beta'(\pi)$ for some $\pi \in \Pi_k$ implies $p = p'$ (hence $\beta = \beta'$) and*
– *$p \notin \beta(\Pi_k), p' \notin \beta'(\Pi_k)$ implies $p \neq p'$.*

Also, assume $E \cap E' = \emptyset$. The concatenation *$M.M'$ of M and M' is the named MSC $((P'', E'', \{\leq''_p\}_{p \in P''}, \lambda'', \mu''), \beta'')$ where*[1]

[1] For two functions h and l over disjoint domains, $h \cup l$ denotes the function over the combined domain defined in the expected manner.

– $P'' = P \cup P'$; $E'' = E \cup E'$
– $\leq''_p = \begin{cases} \leq_p & \text{for } p \in P \setminus \beta(\Pi_k) \\ \leq'_p & \text{for } p \in P' \setminus \beta'(\Pi_k) \\ \leq_p \cup \leq'_p \cup (E_p \times E'_p) & \text{for } p \in \beta(\Pi_k) \end{cases}$
– $\lambda'' = \lambda \cup \lambda'$; $\mu'' = \mu \cup \mu'$; $\beta'' = \beta\ (= \beta')$.

It is easy to see that whenever we substitute named MSCs with equivalent ones, their compositions yield equivalent named MSCs.

The idea of the join composition is to take the union of the processes of two named MSCs and to identify a new set of processes for the process identifiers. Let $\langle \Lambda, \Delta \rangle$ denote that Λ and Δ form a partition of the set of process identifiers Π_k, i.e. $\Lambda \cup \Delta = \Pi_k$ and $\Lambda \cap \Delta = \emptyset$.

Definition 4. *Let $M = ((P, E, \{\leq_p\}_{p \in P}, \lambda, \mu), \beta)$ and $M' = ((P', E', \{\leq'_p\}_{p \in P'}, \lambda', \mu'), \beta')$ be two named MSCs. Because of renaming, we may assume that $P \cap P' = \emptyset$ and $E \cap E' = \emptyset$. Let $\langle \Lambda, \Delta \rangle$ be a partition of Π_k. The* join *of M and M' with respect to $\langle \Lambda, \Delta \rangle$ is denoted by $\mathsf{join}_{\langle \Lambda, \Delta \rangle}(M, M')$ and is the named MSC $((P'', E'', \{\leq''_p\}_{p \in P''}, \lambda'', \mu''), \beta'')$ defined by*

– *$P'' = P \cup P''$; $E'' = E \cup E'$,*
– *$\{\leq''_p\}_{p \in P''}$ is defined by $\leq''_p = \leq_p$ for $p \in P$ and $\leq''_p = \leq'_p$ for $p \in P'$,*
– *$\lambda'' = \lambda \cup \lambda'$; $\mu'' = \mu \cup \mu'$,*
– *$\beta''(\pi)$ yields $\beta(\pi)$ for $\pi \in \Lambda$ and $\beta'(\pi)$ for $\pi \in \Delta$.*

Note that no event of M is causally related to any event of M' in $\mathsf{join}_{\langle \Lambda, \Delta \rangle}(M, M')$. Also, it is obvious that the operation is robust for equivalent named MSCs.

Concatenation and join of named MSCs is illustrated in Figure 1 below.

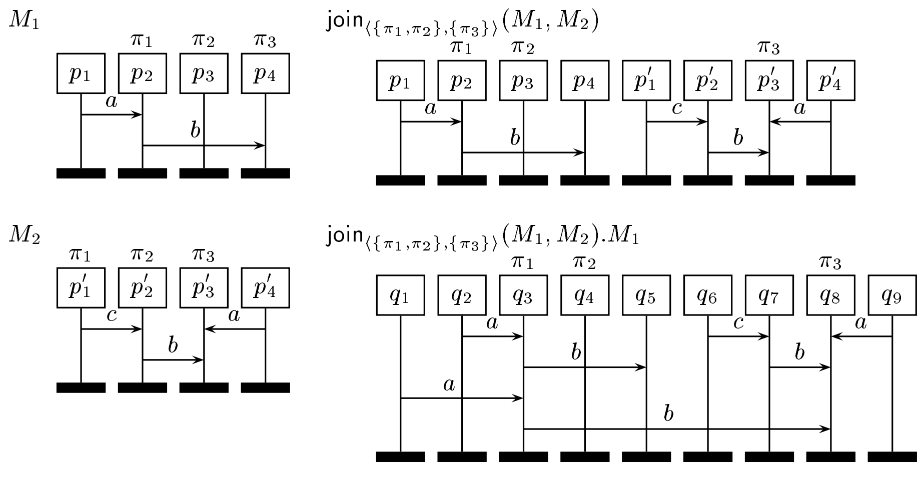

M_1
π_1 π_2 π_3
p_1 p_2 p_3 p_4
a b
$\mathsf{join}_{\langle\{\pi_1,\pi_2\},\{\pi_3\}\rangle}(M_1, M_2)$
p'_1 p'_2 p'_3 p'_4
c
M_2
$\mathsf{join}_{\langle\{\pi_1,\pi_2\},\{\pi_3\}\rangle}(M_1, M_2).M_1$
q_1 q_2 q_3 q_4 q_5 q_6 q_7 q_8 q_9

Fig. 1. Concatenation and join of MSCs

Let $\mathcal{M}$ be a finite set of named MSCs (named with process identifiers Π_k). Let Σ be an alphabet and $\overline{} : \Sigma \to \mathcal{M}$ a bijection between Σ and $\mathcal{M}$. Let us fix these for the rest of the paper.

A *term* is an expression given by:

$$term ::= b \mid term.term \mid \mathsf{split}_{\langle \Lambda, \Delta \rangle}(term, term)$$

where $b \in \Sigma$ and $\langle \Lambda, \Delta \rangle$ is a partition of Π_k.

To any term t, we can now associate a named MSC $[t]$ inductively as follows: $[b] = \bar{b}$, $[t_1.t_2] = [t_1].[t_2]$ and $[\mathsf{split}_{\langle \Lambda, \Delta \rangle}(t_1, t_2)] = \mathsf{join}_{\langle \Lambda, \Delta \rangle}([t_1], [t_2])$.

For example, $b.\mathsf{split}_{\langle \Lambda, \Delta \rangle}(b_1, b_2).b'$ is a term (where $b, b_1, b_2, b' \in \Sigma$). In this term, we start with the MSC $\bar{b}$. Then the processes Λ and Δ split. In the first branch, the processes in Λ interact with a new set of processes (instantiated with processes identifiers in Δ) and interact with them as specified in $\bar{b_1}$. Similarly, in the other branch, new processes with identifiers in Λ are created and interact with the processes labelled Δ in $\bar{b}$, according to $\bar{b_2}$. Then, the original processes join and interact as specified in $\bar{b}'$.

A set of terms hence represents a set of MSCs obtained using concatenation and join operations of the MSCs in $\mathcal{M}$. We specify sets of terms using a grammar:

Definition 5. *Let $\mathcal{N}$ be a finite set of non-terminals and $S_0 \in \mathcal{N}$ a start symbol. A* fork-and-join MSC grammar *is a tuple $(\mathcal{R}, S_0, \Sigma, \mathcal{M}, \overline{})$ where $\mathcal{R}$ is a set of (context-free grammar) rules of the form $S \to nterm$ where nterm is a term with non-terminals, given by the grammar*

$$nterm ::= b \mid S \mid nterm.nterm \mid \mathsf{split}_{\langle \Lambda, \Delta \rangle}(nterm, nterm)$$

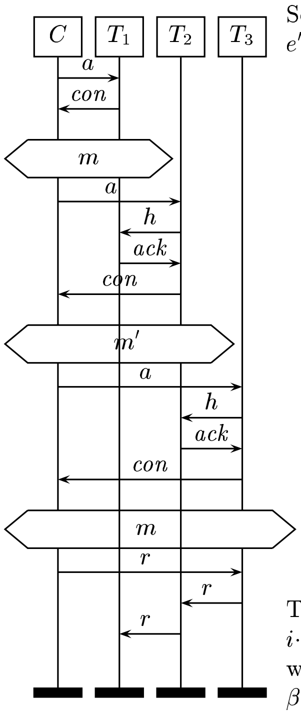
where $b \in \Sigma$, $S \in \mathcal{N}$ and $\langle \Lambda, \Delta \rangle$ is a partition of Π_k.

The set of terms that are derived using a fork-and-join MSC grammar $\mathcal{G}$ is denoted $T(\mathcal{G})$.

Definition 6. *Let $\mathcal{G} = (\mathcal{R}, S_0, \Sigma, \mathcal{M}, \overline{})$ be a* fork-and-join MSC grammar*. The* language *of $\mathcal{G}$ is denoted by $\mathcal{L}(\mathcal{G})$ and is defined as*

$$\mathcal{L}(\mathcal{G}) = \{m \mid \text{there is } t \in T(\mathcal{G}) \text{ and an assignment } \beta \text{ with } (m, \beta) \in [t]\}$$

Example 1. Imagine a scenario in which a car C travels from a source to a destination on a route guided by a number of transmitters. The number of transmitters is not fixed. A typical scenario for three transmitters is depicted in Figure 2. Initially the car C sends an "a" (approach) signal to the first transmitter T_1. On receiving a connect signal "con" from T_1, the car and the transmitter interact using a protocol which is described by some MSC m. As the car moves away from a transmitter T_i and approaches the next transmitter T_{i+1}, it sends the approach signal to T_{i+1}. T_{i+1}, on receiving this, requests T_i to hand-over ("h") the control. When T_{i+1} receives an acknowledgement, it sends a connect signal "con" to C, upon which they start their protocol m. Once the car reaches its destination, it informs its current transmitter and this message is relayed back to the first transmitter. In the figure below, we describe the above scenarios in terms of a fork-join grammar.



Set $k = 3$, $\Pi = \{c, t_1, t_2\}$, and $\Sigma = \{i, m, m', n, n', b, b', e, e', \varepsilon\}$. The names in Σ denote the following named MSCs:

- ε denotes the empty MSC (i.e. the one with no events)
- i is the named MSC where c sends "a" to t_1 and t_1, on receiving this, sends back "con" to c,
- The MSC m is the protocol that the car establishes with the transmitter t_1,
- n describes this sequence of messages: c sends "a" to t_2, t_2 sends "h" to t_1, t_1 sends ack to t_2 and t_2 sends "con" to c,
- b describes t_2 sending "r" to t_1,
- e describes c sending "r" to t_1,
- m',n', b', e' are the MSCs obtained from m, n, b and e, respectively, by switching the roles of t_1 and t_2.

$$\begin{aligned}
\mathcal{S} &\to i \cdot m \cdot \mathcal{L}_0\\
\mathcal{L}_0 &\to n \cdot m' \cdot \mathsf{split}_{\langle\{t_1\},\{c,t_2\}\rangle}(\varepsilon, \mathcal{L}_1) \cdot b\\
\mathcal{L}_0 &\to e\\
\mathcal{L}_1 &\to n' \cdot m \cdot \mathsf{split}_{\langle\{t_2\},\{c,t_1\}\rangle}(\varepsilon, \mathcal{L}_0) \cdot b'\\
\mathcal{L}_1 &\to e'
\end{aligned}$$

This generates, for example, the term $i{\cdot}m{\cdot}n{\cdot}m'{\cdot}\mathsf{split}_{\langle\{t_1\},\{c,t_2\}\rangle}(\varepsilon, n'{\cdot}m{\cdot}\mathsf{split}_{\langle\{t_2\},\{c,t_1\}\rangle}(\varepsilon, e){\cdot}b'){\cdot}b$, which depicts the figure on the left, with $\beta = \{c \mapsto C, t_1 \mapsto T_1, t_2 \mapsto T_2\}$.

Fig. 2. A typical scenario for three transmitters and the grammar generating them

Monadic Second-Order Logic

We assume a supply $eVar = \{x, y, \ldots\}$ of individual (first-order) variables, which are interpreted over events of an MSC, and a supply $EVar = \{X, Y, \ldots\}$ of (second-order) set variables, which are interpreted over sets of events. Since the number of processes is unbounded, we extend our logic to quantification over processes as well. Thus, let $pVar = \{u, v, \ldots\}$ be a supply of (first-order) process variables and $PVar = \{U, \ldots\}$ be a set of (second-order) variables interpreted over sets of processes.

The syntax of monadic second-order logic (MSO) over MSCs is given by

$$\begin{aligned}
\varphi ::= (u, x) \overset{a}{\to} (v, y) \mid x \leq_u y \mid x \in X \mid u \in U \mid\\
\neg\varphi \mid \varphi \vee \varphi \mid \exists x\, \varphi \mid \exists X\, \varphi \mid \exists u\, \varphi \mid \exists U\, \varphi
\end{aligned}$$

where $u, v \in pVar$, $U \in PVar$, $x, y \in eVar$, $X \in EVar$, and $a \in \Gamma_M$. The notions of free and bound variables are introduced as usual.

The intuitive meaning of $(u, x) \overset{a}{\to} (v, y)$ is that x is a send-event of process u and y is the corresponding receive event in process v and the message sent is a. $x \leq_u y$ holds iff y follows x in the total order of the process given by u. Let $m = (P, E, \{\leq_p\}_{p \in P}, \lambda, \mu)$ be an MSC. Let $\mathcal{I}$ be an interpretation function which assigns to every x, X, u and U, an event, a set of events, a process and a set of processes, respectively. The satisfaction relation $m \models_{\mathcal{I}} \varphi$ for a formula $\varphi \in$ MSO is inductively defined as follows:

– $m \models_{\mathcal{I}} (u, x) \xrightarrow{a} (v, y)$ iff $\lambda(\mathcal{I}(x)) = (\mathcal{I}(u)!\mathcal{I}(v), a)$, $\lambda(\mathcal{I}(y)) = (\mathcal{I}(v)?\mathcal{I}(u), a)$, and $\mu(\mathcal{I}(x)) = \mathcal{I}(y)$,
– $m \models_{\mathcal{I}} x \leq_u y$ iff $\mathcal{I}(x) \leq_{\mathcal{I}(u)} \mathcal{I}(y)$.

The remaining constructs are defined as usual. For a formula without free variables, we write $m \models \varphi$ instead of $m \models_{\mathcal{I}} \varphi$. Note that the causal order relation $\leq$, though not explicitly present, can be expressed using our syntax (see [4]). Also, the logic cannot distinguish between equivalent MSCs.

We can now state the main result of the paper:

Theorem 1. *The problem of checking, given a fork-and-join MSC grammar $\mathcal{G}$ and an MSO formula φ, whether all the MSCs corresponding to terms in $T(\mathcal{G})$ satisfy φ, is decidable.*

3 Model Checking

Trees and Automata: A *tree* is a graph $T = (N, Edg)$ where N is a finite subset of $\{0, 1\}^*$ of nodes such that (i) N is prefix-closed and (ii) if $x \in N$, then either both $x.0$ and $x.1$ are in N or neither is in N, and $Edg = \{(x, x.i) \mid x, x.i \in N, i \in \{0, 1\}\}$. Thus the trees we consider are finite binary trees where every node has either zero or two children.

For a finite set of labels Γ, a Γ-labelled tree is a pair (T, λ) where $T = (N, Edg)$ is a tree and $\lambda : N \to \Gamma$ is a labelling function that assigns a label to every node of the tree.

A tree-automaton over Γ-labelled trees is a tuple $\mathcal{A} = (Q, init, \delta, Q_f)$ where Q is a finite set of states, $init : \Gamma \to 2^Q$ is a function that associates a set of initial states with every label, $Q_f \subseteq Q$ is a set of final states and $\delta : Q \times Q \times \Gamma \to 2^Q$ is a bottom-up transition function that associates a set of possible states to a node depending on its label and the states associated with its children.

A run of such an automaton over a Γ-labelled tree (T, λ), with $T = (N, Edg)$ is a function $\rho : N \to Q$ such that for every leaf v in T, $\rho(v) \in init(\lambda(v))$, and for every internal node v, $\rho(v) \in \delta(\rho(v.0), \rho(v.1), \lambda(v))$. Such a run is said to be accepting if $\rho(\epsilon) \in Q_f$, i.e. if the root is labelled by a final state. A labelled tree is accepted by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ over it. The language of trees accepted by $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$, is the set of trees it accepts. A set of Γ-labelled trees is said to be regular if there is an automaton whose language is this set.

Parse Trees and i-Trees: We work with trees that represent the parse-trees of terms. Formally, for a given alphabet Σ, the set of *parse trees* over Σ is the set of all Γ-labelled trees, where $\Gamma = Ops \cup \Sigma$ with $Ops = \{.\} \cup \bigcup_{\langle \Lambda, \Delta \rangle} \{\mathsf{split}_{\langle \Lambda, \Delta \rangle}\}$, in which all the leaves are labelled with letters in Σ and all the internal nodes are labelled with letters in Ops.

For a term t over Σ, we can clearly associate a unique parse tree over Σ. Also, every parse tree over Σ corresponds to a term over Σ. It is also easy to see that the set of parse trees over Σ is regular. We assume henceforth that the automata we construct work only over parse trees—this is an acceptable assumption since we can construct an automaton that accepts the set of all parse trees and take its intersection with the automata we construct.

In addition to having a tree represent a term, we work with trees that represent terms along with an interpretation of a set of variables. Let V be a finite set of first-order and second-order process and event variables. Recall that every $b \in \Sigma$ is associated with an MSC $\bar{b}$. A *partial interpretation* of V over b is a function I that maps some first-order (second-order) process variables in V to a process (a set of processes) in $\bar{b}$, and some first-order (second-order) event variables to an event (a set of events) in $\bar{b}$. In other words, it is an interpretation of some subset of variables $V' \subseteq V$ over $\bar{b}$. An *interpretation tree*, or an i-tree over V is a Γ-labelled tree where $\Gamma = Ops \cup \{(b, I) \mid b \in \Sigma, I \textit{ is a partial interpretation of } V \textit{ over } b\}$. We call the underlying term the tree represents as the term associated with the i-tree, i.e. the term associated with the parse tree obtained when each leaf labelled (b, I) is instead labelled b.

If an i-tree over V is associated with a term t, then the i-tree is supposed to represent both t and an interpretation for the variables in V to the processes and events in $[t]$, the MSC associated with t. For an interpretation function $\mathcal{I}$ of V over $[t]$, it is clear how to associate an i-tree that represents it: For each leaf in the parse tree of t labelled b, map a process variable p to a process in $\bar{b}$, provided the process is the same process interpreted by $\mathcal{I}$ in $[t]$. Similarly, process set variables, event and event set variables can be interpreted locally at each leaf. We refer to this i-tree as the one that corresponds to $\mathcal{I}$. Note that in this i-tree, a first-order event variable gets interpreted at only one leaf. However, a first-order process variable could get interpreted at many leaves—this is because a process in $[t]$ is formed using events of many MSCs at the leaves of the i-tree.

It is clear from the above that not every i-tree over V which is associated with a term t may correspond to an interpretation $\mathcal{I}$ over V—in particular, the first-order process variables defined at various leaves may not correspond to a single process in $[t]$. Also, we clearly require that a first-order event variable is given an interpretation in only one of the leaves. We say an i-tree is *legal* if there is an interpretation $\mathcal{I}$ over V that it corresponds to. Our main task now is to identify the set of legal i-trees.

Let us add an additional layer of labelling to the nodes of an i-tree as follows. The labelling set will be the set Ξ where each element of Ξ is of the form $(\mathrm{IP}, \mathrm{IE}, \eta, \zeta)$ where:

- IP ("interpreted process variables") and IE ("interpreted event variables") are subsets of first-order process and event variables of V, respectively,
- $\eta \subseteq \mathrm{IP} \times \{U \mid U \textit{ is a second-order process variable in } V\}$,
- $\zeta : \mathrm{IP} \rightharpoonup \Pi_k$ is a *partial* function that associates some first-order process variables of IP to process identifiers.

When we label a node v of an i-tree with $(\mathrm{IP}, \mathrm{IE}, \eta, \zeta)$, the intuition is that in the subterm represented by the sub-tree rooted at v, IP and IE are the set of first-order process and event variables that have been given an interpretation in the MSC associated with the sub-term. Also, $(u, U) \in \eta$ iff in the interpretation of u in the sub-term, the process u has been declared to be an element of U. The meaning of ζ is that $\zeta(u) = \pi_i$ iff in the named MSC corresponding to the subterm, the process identifier π_i is assigned the same process as the one u is interpreted with.

The labelling of leaves of an i-tree is straightforward—we label a leaf labelled (b, I) with $(\mathrm{IP}, \mathrm{IE}, \eta, \zeta)$ where IP and IE are the set of all first-order process variables and first-order event variables interpreted by I, respectively; η is the set of all (u, U) where $u \in \mathrm{IP}$ and the process associated with u belongs to the set of processes associated with U, and $\zeta(u) = \pi_i$ iff u is interpreted as a process with identifier π_i.

We can now label the tree bottom-up, starting at the leaves. If v is a node of the tree and its children $v.i$ are labelled $(\mathrm{IP}_i, \mathrm{IE}_i, \eta_i, \zeta_i)$, where $i \in \{0, 1\}$, then we can label v as follows. There are two cases, depending on whether the node v is labelled '.' or '$\mathsf{split}_{\langle \Lambda, \Delta \rangle}$'.

If v is labelled '.', then we say that the labels of its children are consistent if (i) for every $u \in \mathrm{IP}_0 \cap \mathrm{IP}_1$ and $U \in V$, $(u, U) \in \eta_0$ iff $(u, U) \in \eta_1$ and (ii) for every $u \in \mathrm{IP}_0 \setminus \mathrm{IP}_1$, ζ_0 is undefined on u and for every $u \in \mathrm{IP}_1 \setminus \mathrm{IP}_0$, ζ_1 is undefined on u, (iii) for every $u \in \mathrm{IP}_0 \cap \mathrm{IP}_1$, both ζ_0 and ζ_1 are defined on u and $\zeta_0(u) = \zeta_1(u)$ and (iv) $\mathrm{IE}_0 \cap \mathrm{IE}_1 = \emptyset$.

Intuitively, (i) says that, since the MSCs associated with the children of v are going to be concatenated, if a process variable is defined in both MSCs, then the sets of process set variables they are declared to belong to must be the same. Conditions (ii) and (iii) say that for every interpreted process variable u, either u is interpreted only in one of the named MSCs and is not associated with any process identifier, or u is interpreted in both named MSCs and are associated with the same process identifier. The reason behind these conditions is immediate when one observes that when two named MSCs are concatenated, processes whose event-lines get concatenated are exactly those which are associated with process identifiers. The last condition ensures that a first-order event variable is not interpreted in both the subtrees.

If v's children are consistent, it is clear that we can label v as $(\mathrm{IP}, \mathrm{IE}, \eta, \zeta)$ where $\mathrm{IP} = \mathrm{IP}_0 \cup \mathrm{IP}_1$, $\mathrm{IE} = \mathrm{IE}_0 \cup \mathrm{IE}_1$, $\eta = \eta_0 \cup \eta_1$ and $\zeta(u) = \zeta_0(u)$ if $u \in \mathrm{IP}_0$ and $\zeta_1(u)$, otherwise.

Let us now turn to the case when v is labelled '$\mathsf{split}_{\langle \Delta_0, \Delta_1 \rangle}$'. We say that the labels of its children are consistent if $\mathrm{IP}_0 \cap \mathrm{IP}_1 = \emptyset$ and $\mathrm{IE}_0 \cap \mathrm{IE}_1 = \emptyset$. Hence, all that we require is that the first-order process and event variables interpreted in the component MSCs be disjoint. If the labels of v's children are consistent, then we can label v as $(\mathrm{IP}, \mathrm{IE}, \eta, \zeta)$, where $\mathrm{IP} = \mathrm{IP}_0 \cup \mathrm{IP}_1$, $\mathrm{IE} = \mathrm{IE}_0 \cup \mathrm{IE}_1$, $\eta = \eta_0 \cup \eta_1$ and ζ is given as follows: for every $u \in \mathrm{IP}_i$, if $\zeta_i(u) \in \Delta_i$, $\zeta(u) = \zeta_i(u)$, where $i \in \{0, 1\}$. It is easy to see that this update maintains the semantics of the labelling according to the join operation of the two MSCs.

We say a labelling of the nodes of an i-tree is consistent if it can be labelled using the above rules (i.e. every leaf is labelled as described above and every internal node's children are labelled consistently and respects the above labelling rule). It is now easy to observe the following:

Proposition 1. *An i-tree over V is legal iff it has a consistent labelling in which the label of the root is (IP, IE, η, ζ) with $IP \cup IE$ spanning all the first-order process and event variables of V.*

The following is now immediate:

Lemma 1. *The set of all legal i-trees is regular.*

Proof. The labelling set Ξ above can be taken to be the set of the states of a tree-automaton working over i-trees. It is easy to engineer such an automaton that accepts an i-tree iff the i-tree admits a consistent labelling. $\square$

In the sequel, we assume that a formula which uses a variable, has at most one quantification involving it. In the style of the well-known automata theoretic approach to decidability [9], we can now show:

Lemma 2. *For any MSO formula φ with free-variables V, the set of all legal i-trees t over V such that the MSC represented by t satisfies φ under the interpretation of V defined by t, is regular.*

Proof. The proof will be by induction on the structure of the formula φ:

For the atomic formula of the kind $(u, x) \xrightarrow{a} (v, y)$, when reading a legal i-tree, the automaton can check the formula by checking if there is some leaf labelled (b, I), where both x and y are interpreted by I, and u and v are interpreted as the processes x and y belong to, respectively, and x and y are matching send and receive events of a message labelled a in $[b]$. It is easy to construct an automaton which accepts a tree iff it has such a leaf.

For the atomic formula $x \leq_u y$, the automaton is more complex. Recall the labelling used in defining the consistency of an i-tree. It is easy to verify that $x \leq_u y$ iff x and y are both interpreted (at leaves) as events of a process that is interpreted as u (in the leaves) and one of the following hold:

– there is a leaf (say labelled (b, I)) where both x and y are interpreted and the event interpreted for x is causally before the event interpreted for y in $[b]$, or
– Let the bottom-most internal node of the tree which is labelled $(\mathrm{IP}, \mathrm{IE}, \eta, \zeta)$ with $x, y \in \mathrm{IE}$ be v and let its children $v.i$ be labelled $(\mathrm{IP}_i, \mathrm{IE}_i, \eta_i, \zeta_i)$. Then v is labelled '.' and $x \in \mathrm{IE}_0$ and $y \in \mathrm{IE}_1$.

Intuitively, $x \leq_u y$ iff they belong to the process interpreted as u and if they are both interpreted at a leaf and they are causally related in the MSC defined at the leaf, or they get causally related by a concatenation operation. We can design an automaton which finds the consistent labelling and accepts the tree iff the above property is satisfied.

The atomic formula "$x \in X$" can be checked easily by checking them at the leaf where x is interpreted (say labelled (b, I)) and checking if $I(x) \in I(X)$. The formula "$u \in U$" can be handled similarly.

For the formula $\neg\varphi$, we can take the automaton for φ, complement it and take its intersection with the automaton accepting all legal i-trees. For formulas formed using disjunction, $\varphi \vee \varphi'$, we can take the automata for φ and φ', tinker with them if necessary so that they now work on i-trees over the free variables in $\varphi \vee \varphi'$, and then construct an automaton that accepts the union of these two tree languages.

The formulas formed with existential quantification of the form $\exists W \varphi(W)$ (where W is a first- or second-order, event or process variable) can be handled by taking the automaton for $\varphi(W)$ (call this $\mathcal{A}_\varphi$) and then building an automaton accepting i-trees over V' (where V' is the set of free variables of φ but with W removed from it). This automaton first guesses an appropriate extension of the interpretation at the leaves to include an interpretation of W and then proceeds to simulate $\mathcal{A}_\varphi$ on this extended labelled tree. It hence accepts a legal i-tree over V' iff there is a legal i-tree over $V' \cup \{W\}$ that extends the labelling to include an interpretation of W, and this tree is accepted by $\mathcal{A}_\varphi$. $\square$

For a sentence φ (without free variables), the i-trees are over the empty set and hence are isomorphic to parse-trees. Invoking the above lemma, we have:

Theorem 2. *For any MSO sentence φ, the set of all parse-trees over Σ that correspond to terms that satisfy φ is regular. Moreover, one can effectively construct an automaton $\mathcal{A}_\varphi$ that accepts these trees.*

In the model-checking problem, we are given a fork-and-join MSC grammar $\mathcal{G} = (\mathcal{R}, S_0, \Sigma, \mathcal{M}, \overline{})$ and an MSO formula φ. The following is easy to observe:

Lemma 3. *The set of all parse trees corresponding to terms in $T(\mathcal{G})$ is regular.*

Proof. One can first rewrite the rules in $\mathcal{G}$ (but preserving the language of terms) such that each rule is of the form $S \to b$, $S \to T$, $S \to nterm.nterm$ or $S \to \mathsf{split}_{\langle \Lambda, \Delta \rangle}(nterm, nterm)$. (This may require more nonterminals.) Then one can show that a parse tree of a term belongs to $T(\mathcal{G})$ iff there is a labelling of the internal nodes with nonterminals such that the root is labelled with S_0 and if a node is labelled S, then the label of its children along with the parse tree label of the node is according to some rule in the grammar. It is easy to build such an automaton which checks whether there is such a labelling. $\square$

To solve the model-checking problem, we first construct, using Lemma 2, an automaton $\mathcal{A}_{\neg\varphi}$ that accepts precisely the parse-trees over Σ which correspond to MSCs that satisfy $\neg\varphi$. We also construct an automaton $\mathcal{A}_\mathcal{G}$ that accepts the parse trees of terms in $T(\mathcal{G})$ (using Lemma 3). The problem then boils down to checking whether $\mathcal{L}(\mathcal{A}_{\neg\varphi}) \cap \mathcal{L}(\mathcal{A}_\mathcal{G})$ is nonempty, which is decidable. This establishes Theorem 1.

4 Discussion

We have presented a formalism to specify languages of MSCs over unboundedly many processes, and shown that MSO-model checking is decidable for this class. Note that there have been similar efforts to extend model-checking for systems with a fixed number of processes to unboundedly many processes in the area of verification of *parameterized systems*. In that setting, however, it turns out that the model-checking problem gets quickly undecidable and while there are many efforts to find decidable fragments, there is no formalism such that all problems expressed in the formalism admit an effective model-checking procedure. In this light, the fact that there is a decidable formalism for specifying scenarios for unboundedly many processes is interesting.

It is easy to see that MSGs can be modelled in our framework even without the split operator. However, even discarding the split, our framework can be seen to be more powerful than MSGs, as it allows a context-free grammar to describe the concatenations of the atomic MSCs.

A number of extensions are worthy of study. First, there is an extension of MSGs, called compositional MSGs, where in the specification of an MSC, a send-event can be defined without a matching receive—the matching receive can be defined after an arbitrary delay. Model-checking for MSGs equipped with this feature (CMSGs) specifications is known to be decidable [6]. It turns out that we can extend our results to the class where the atomic named MSCs $\mathcal{M}$ are allowed to have such unmatched send-events. Also, MSO logic can be enriched with modulo counting quantifiers, preserving decidability. See [4] for details.

A future direction is to define structural temporal logics for unboundedly many processes and adapting our procedures to that fragment in order to yield interesting yet efficient algorithms for model-checking. Extensions of the presented formalism to handle infinite MSCs, aimed at analyzing liveness properties, would also be interesting.

References

1. R. Alur and M. Yannakakis. Model checking of message sequence charts. In *Proc. 10th Int'l. Conf. on Concurrency Theory*, LNCS 1664, p. 114–129. Springer, 1999.
2. D. Caucal. On infinite transition graphs having a decidable monadic theory. In *Proc. the 23th International Colloquium on Automata, Languages and Programming (ICALP'96)*, LNCS 1099, p. 194–205, Springer, 1996.
3. B. Courcelle. On the expression of graph properties in some fragments of monadic second-order logic. In *Descriptive complexity and finite models*, volume 31.DIMACS Series in Discrete Mathematics and Theoretical Computer Sciences, June 1997.
4. M. Leucker, P. Madhusudan, and S. Mukhopadhyay. Dynamic message sequence charts. Technical Report MS-CIS-02-27, University of Pennsylvania, 2002.
5. P. Madhusudan. Reasoning about sequential and branching behaviours of message sequence graphs. In *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP'01)*, LNCS 2076, p. 396–407. Springer, 2001.
6. P. Madhusudan and B. Meenakshi. Beyond message sequence graphs. In *Proc. 21st Conference on Foundations of Software Technology and Theoretical Computer Science*, LNCS 2245, p. 256–267. Springer, 2001.
7. R. Milner. *A Calculus for Communicating Processes*, LNCS 92, Springer, 1980.
8. ITU-TS. ITU-TS Recommendation Z.120: Message Sequence Chart 1996 (MSC96). Technical report, ITU-TS, Geneva, 1996.
9. W. Thomas. Languages, automata and logic. In *Handbook of Formal Languages*, volume 3, Beyond Words. Springer, 1997.

The Complexity of Compositions of Deterministic Tree Transducers

Sebastian Maneth

LIACS, Leiden University
P.O.Box 9512, 2300 RA Leiden, The Netherlands
`maneth@liacs.nl`

Abstract. Macro tree transducers can simulate most models of tree transducers (e.g., top-down and bottom-up tree transducers, attribute grammars, and pebble tree transducers which, in turn, can simulate all known models of XML transformers). The string languages generated by compositions of macro tree transducers (obtained by reading the leaves of the output trees) form a large class which contains, e.g., the IO hierarchy and the EDT0L control hierarchy. Consider an arbitrary composition τ of (deterministic) macro tree transducers. How difficult is it, for a given input tree s, to compute the translation $t = \tau(s)$? It is shown that this problem can be solved (on a RAM) in time linear in the sum of the sizes of s and t. Moreover, the problem to determine, for a given t of size n, whether or not there is an input tree s such that $t = \tau(s)$ is in DSPACE(n); this means that output languages of compositions of macro tree transducers are deterministic context-sensitive. The involved technique of compressing intermediate results of the composition, also gives a new proof of the fact that the finiteness problem for τ's range is decidable.

Keywords: Deterministic Macro Tree Transducers, Complexity

1 Introduction

Macro tree transducers (MTTs) [EV85,CF82,FV98] are a well-known model of syntax directed translation. They can be obtained by adding parameters (used as in a macro grammar [Fis68]) to top-down tree transducers. Recently, tree transducers have gained new interest, as a model of XML query languages [Via01]. In particular, the k-pebble tree transducer [MSV00] has been proposed as a general framework which captures all known XML query languages. As it turns out, pebble tree transducers can be simulated by compositions of macro tree transducers [EM]. In this paper we only consider total deterministic transducers. Then, another well-known model of translations that can be simulated by MTTs is the attribute grammar [Knu68]. Attribute grammars have several nice features concerning complexity: they can be evaluated in linear time, assuming that each semantic rule is evaluated in constant time (see, e.g., [DJL88]). From now on, consider an attribute grammar (AG) as tree transducer [Fül81]: the input is a derivation tree of the underlying context-free grammar, and the output is the (uninterpreted) output term. If the output term is represented as a "term graph", i.e., a tree in which sharing of common subtrees is allowed, then each semantic rule can be evaluated in constant time, and therefore the (derivation) tree to term graph translation realized by an AG can be evaluated in time linear in the size of the input tree s on a Random Access Machine (RAM). The unfolding of a term graph into the output tree t can be done (on a RAM) in time linear in the size of t. Altogether, the tree to tree translation of an AG can be done on a RAM in time $O(m)$ where $m = |s| + |t|$. Since AGs cannot realize all macro tree translations, they can be thought of as an efficient implementation of a subclass of the macro tree translations.

If we consider the composition of tree transducers, then the situation changes drastically: For an MTT M we can construct two attribute grammars such that their composition equals M's translation (see, e.g., Chapter 6 of [FV98]). Hence, a composition of MTTs can always be implemented by a composition of AGs. But, how efficient can a composition τ of AGs be evaluated? Here the problem arises, that the term graph generated by the first AG of τ, has to be unfolded (into an "intermediate result") before the next AG can translate it (note that the size of such an output tree can be exponential in the size of the input tree). Hence the question arises, how large are the intermediate results which are needed to compute $\tau(s)$? To answer this question, we work on certain restricted MTTs which can be simulated by AGs, and show that they can be made "productive" in such a way that the size of an, appropriately "compressed" input tree is linear bounded by the size of the output tree. Since the compression of the input tree can be incorporated in the previous transducer, the size of each intermediate result will be linear bounded by the size of $\tau(s)$. Hence, even the composition of tree transducers can be computed (on a RAM) in time linear in the sum of the sizes of s and $\tau(s)$. MTTs are also particular functional programs [Vog91], and their composition with particular subclasses (of MTTs) can be seen [Küh98] as a method of deforestation [Wad90]. Hence, our result is related: in [Küh98] intermediate results are avoided (by composition), and we keep (and even introduce) intermediate results but limit their size.

Let us discuss in more detail how to make an MTT productive (Section 3). Consider a composition $M_1 \circ \cdots \circ M_k$ of MTTs (which means, first M_1 translates an input tree, then M_2 takes M_1's output as input, and so on). We show that every MTT can be decomposed into a top-down tree transducer (which "compresses" the input tree) followed by an MTT that is productive, i.e., which uses each node of its input tree in order to generate output. Productivity guarantees that the size of the input tree is linearly bounded by the size of the corresponding output tree. Thus, M_k can be decomposed into the top-down tree transducer C, followed by the productive MTT M_k'. Hence, if t is the output of M_k, then M_k' has an input tree s that generates t, and the size of s is linearly bounded by the size of t. Clearly, the transducer C is not productive: it removes all useless nodes from an input tree. However, due to the fact that C is a top-down tree transducer, it can be composed with M_{k-1} [EV85], i.e., $M_{k-1} \circ C$ can be realized by a new MTT M_{k-1}^C. Intuitively, M_{k-1}^C no longer generates input trees for M_k (which might have contained many useless nodes), but generates the corresponding compressed input trees for the productive M'_k. The main result is Theorem 12: Any composition of MTTs can be realized by a composition of a linear top-down tree transducer C, followed by the composition of productive MTTs (to be precise, C is preceded by a finite state relabeling).

In Section 4, Theorem 12 is used to prove the first complexity result: Computing $\tau(s)$, for a composition τ of MTTs, can be done in time linear in the sum of the sizes of s and $\tau(s)$. Finally, we treat output languages; for τ and a regular tree language R (i.e., an "input type" R) we consider the problem: Given an output tree t, is there an input tree s in R such that $\tau(s) = t$? (the "word problem for τ's output".) It is proved that this problem can be decided in DSPACE(n), i.e., we construct a deterministic Turing Machine that uses space linear in the size n of t and decides whether or not there is an $s \in R$ with $t = \tau(s)$. The proof is based on Theorem 12, and on the result of [Eng86] that output languages of attribute grammars are in LOG(CFL). Note that composition of MTTs gives rise to a proper hierarchy of output languages [EM02]. In [EV85] it was proved that the languages in this hierarchy are recursive. In the case of nondeterministic MTTs we do not know yet the complexity of the corresponding output languages (for nondeterministic top-down tree transducers they are in DSPACE(n) [Bak78]).

Convention: All lemmas and theorems in this paper are effective.

2 Preliminaries

For $k \in I\!N$ let $[k] = \{1, \ldots, k\}$. For strings $v, w_1, \ldots, w_n \in A^*$ and distinct $a_1, \ldots, a_n \in A$, we denote by $v[a_1 \leftarrow w_1, \ldots, a_n \leftarrow w_n]$ the result of (simultaneously) substituting w_i for every occurrence of a_i in v. For functions $f : A \to B$ and $g : B \to C$ their composition is $(f \circ g)(x) = g(f(x))$; note that the order is nonstandard. For sets of functions F and G their composition is $F \circ G = \{f \circ g \mid f \in F, g \in G\}$, and $F^n = F \circ \cdots \circ F$ (n times, $n \geq 1$). Furthermore, F^* denotes the union $\bigcup_{n \geq 1} F^n$. For a class $\mathcal{L}$ of languages, $F(\mathcal{L}) = \{f(L) \mid f \in F, L \in \mathcal{L}\}$ with $f(L) = \{f(x) \mid x \in L\}$.

We assume the reader to be familiar with trees, tree automata, and tree translations (see, e.g., [GS84]). A set Σ together with a mapping rank: $\Sigma \to I\!N$ is called a *ranked* set. For $k \geq 0$, $\Sigma^{(k)}$ is the set $\{\sigma \in \Sigma \mid \text{rank}(\sigma) = k\}$; we also write $\sigma^{(k)}$ to denote that $\text{rank}(\sigma) = k$. For a set A, $\langle \Sigma, A \rangle$ is the ranked set $\{\langle \sigma, a \rangle \mid \sigma \in \Sigma, a \in A\}$ with $\text{rank}(\langle \sigma, a \rangle) = \text{rank}(\sigma)$. The set of all (ordered, ranked) trees over Σ is denoted T_Σ. For a tree t, $V(t)$ is the set of nodes of t, and for a node $u \in V(t)$, $t[u]$ is the label of u in t. For a set A, $T_\Sigma(A)$ is the set of all trees over $\Sigma \cup A$, where all elements in A have rank zero. We fix the *set of input variables* as $X = \{x_1, x_2, \ldots\}$ and the *set of parameters* as $Y = \{y_1, y_2, \ldots\}$. For $k \geq 0$, $X_k = \{x_1, \ldots, x_k\}$ and $Y_k = \{y_1, \ldots, y_k\}$. For a tree $t \in T_\Sigma$, yt denotes the *yield of* t, i.e., the string in $(\Sigma^{(0)} - \{e\})^*$ obtained by reading the leaves of t in pre-order, omitting nodes labeled by the special symbol e (which denotes the empty string). For a class $\mathcal{L}$ of tree languages, $y\mathcal{L}$ denotes $\{yL \mid L \in \mathcal{L}\}$ with $yL = \{yt \mid t \in L\}$. The class of *regular tree languages* is denoted by REGT.

Definition 1. A (total, deterministic) *macro tree transducer* (MTT) is a tuple $M = (Q, \Sigma, \Delta, q_0, R)$, where Q is a ranked alphabet of *states*, Σ and Δ are ranked alphabets of *input* and *output symbols*, respectively, $q_0 \in Q^{(0)}$ is the *initial state*, and R is a finite set of *rules*. For every $q \in Q^{(m)}$ and $\sigma \in \Sigma^{(k)}$ with $m, k \geq 0$ there is exactly one rule of the form $\langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1, \ldots, y_m) \to \zeta$ in R, where $\zeta \in T_{\langle Q, X_k\rangle \cup \Delta}(Y_m)$; the tree ζ is denoted by $\mathrm{rhs}_M(q, \sigma)$. □

A rule as in the definition above is called (q, σ)-rule, and sometimes also q-rule or σ-rule. The rules of M are used as term rewriting rules in the usual way. The derivation relation of M (on $T_{\langle Q, T_\Sigma\rangle \cup \Delta}$) is denoted by $\Rightarrow_M$ and the *translation realized by M*, denoted τ_M, is the total function $\{(s, t) \in T_\Sigma \times T_\Delta \mid \langle q_0, s\rangle \Rightarrow_M^* t\}$. The class of all translations which can be realized by MTTs is denoted by MTT. The MTT M is *linear*, if the right-hand side of every rule is linear in the input variables X_k. The corresponding class of translations is denoted by LMTT. A (total, deterministic) *top-down tree transducer* is an MTT all states of which are of rank zero. The class of all translations which can be realized by top-down tree transducers is denoted by T, and by LT for linear top-down tree transducers. The following example is also used in the proof of Lemma 7 of [EM02].

Example 2. Let Σ be a ranked alphabet with $e \in \Sigma^{(0)}$. Define $M = (\{q_0^{(0)}, q^{(1)}\}, \Sigma, \Delta, q_0, R)$ with $\Delta = \{a^{(1)} \mid a \in \Sigma^{(0)}, a \neq e\} \cup \{e^{(0)}\}$. For every $\sigma \in \Sigma^{(k)}, k \geq 1$, let the rules

$$\begin{aligned} \langle q_0, \sigma(x_1, \ldots, x_k)\rangle \quad &\to \langle q, x_1\rangle(\langle q, x_2\rangle(\ldots(\langle q, x_k\rangle(e))\ldots)) \\ \langle q, \sigma(x_1, \ldots, x_k)\rangle(y_1) &\to \langle q, x_1\rangle(\langle q, x_2\rangle(\ldots(\langle q, x_k\rangle(y_1))\ldots)) \end{aligned}$$

be in R, for every $a \in \Sigma^{(0)} - \{e\}$ let $\langle q_0, a\rangle \to a(e)$ and $\langle q, a\rangle(y_1) \to a(y_1)$ be in R, and let $\langle q_0, e\rangle \to e$ and $\langle q, e\rangle(y_1) \to y_1$ be in R. Then M computes the yield of a tree, seen as a monadic tree. For instance, for $\Sigma = \{\sigma^{(2)}, a^{(0)}, b^{(0)}, c^{(0)}\}$: $\langle q_0, \sigma(a, \sigma(b, c))\rangle \Rightarrow_M \langle q, a\rangle(\langle q, \sigma(b, c)\rangle(e)) \Rightarrow_M^* a(\langle q, b\rangle(\langle q, c\rangle(e))) \Rightarrow_M^* a(b(c(e)))$.

3 Productive Tree Transducers

Since a tree transducer can delete parts of its input tree s, the size of s might be much larger than the size of the corresponding output tree. As discussed in the Introduction, our aim is to bound the size of the *intermediate results*, needed to compute a composition $\tau_{M_1} \circ \cdots \circ \tau_{M_k}(s)$ of MTTs, i.e., the input trees of $M_2, \ldots, M_k$. In fact, these MTTs can be changed in such a way that they are *productive*, i.e., each node of an input tree contributes to the output tree, i.e., it is neither deleted nor “skipped”. As will be proved in Lemma 11, for a productive transducer M the size of each input tree s is linear bounded by the size of the output tree $\tau_M(s)$. We now define this property and then show in the following subsections how to make a transducer productive.

Definition 3. A function τ (on trees) has *linear bounded input* (*by the constant* $c \geq 0$) if, for every $(s,t) \in \tau$, the size of s is linear bounded by the size of t, i.e., $|s| \leq c|t|$. The class of all functions that have linear bounded input is denoted LBI. A transducer has linear bounded input if its translation has. □

Clearly, the class LBI is closed under composition; thus, if τ_i has linear bounded input by c_i, then $\tau_1 \circ \cdots \circ \tau_k$ has linear bounded input by $c = c_1 c_2 \cdots c_k$.

Deletion may be caused by the parameters and by the input variables. We first treat parameters, and then input variables.

3.1 Parameter Nondeleting and Nonerasing MTTs

Here we consider two different types of deletion that are caused by the parameter facility of an MTT. Consider that the rule

$$\langle q, \gamma(x_1)\rangle(y_1, y_2) \to \gamma(\langle q', x_1\rangle(y_2)) \quad \text{“deletion” of } y_1$$

is applied in a derivation. Then the actual parameter tree for y_1 will be deleted. Next, consider that the rule

$$\langle q, \alpha\rangle(y_1) \to y_1 \quad \text{“erasure” of } \alpha$$

is applied. Then no output is produced for the corresponding input node labeled α. Since our aim is to obtain an MTT of linear bounded input, we forbid for productive MTTs both types of deletion shown.

If each parameter y_j, $j \in [m]$, occurs in the right-hand side of a q-rule r where m is the rank of q, then r is parameter nondeleting. If every rule of M is parameter nondeleting, then M is *parameter nondeleting*. The MTT M is *nonerasing*, if no right-hand side of its rules consists of a single parameter only.

The following lemma was proved as Lemma 7.11 of [EM99], in the setting of MTTs with regular look-ahead. It is easy to see that the construction presented there preserves linearity (of input variables). Since regular look-ahead can be simulated by a (total deterministic bottom-up) relabeling (see, e.g., Corollary IV.6.5 of [GS84]), we can state the lemma as follows, where $\mathrm{D_tQRELAB}$ denotes the class of total deterministic bottom-up finite state relabelings (cf., e.g., [Eng75]).

Lemma 4. (Lemma 7.11 of [EM99]) Let M be an MTT. There is a $\mathrm{D_tQRELAB}$ B and an MTT M' such that (1) $\tau_B \circ \tau_{M'} = \tau_M$ and (2) M' is parameter nondeleting and nonerasing. If M is linear, then so is M'.

3.2 Removing Deletion of Input Variables

Consider an MTT M that is parameter nondeleting and nonerasing. What can cause an input tree s to be bigger than $\tau_M(s)$? There are two reasons; first, the deletion of input variables which is dealt with in this subsection, and second the skipping of monadic nodes, discussed in Subsection 3.3. Consider, for example, that the rule

$$\langle q_0, \sigma(x_1, x_2)\rangle \to \gamma(\langle q, x_2\rangle) \qquad \text{“deletion” of } x_1$$

is applied to an input tree $\sigma(s_1, s_2)$. Then the tree s_1 is deleted, i.e., it is never translated by M. One way of guaranteeing that each node of the input tree is translated, is to require that in the right-hand side of every rule each x_i occurs. Instead of this strong static requirement (which is the classical notion of “nondeletion”), we now introduce a more liberal, dynamic restriction (called nondeleting) which also guarantees that each node of the input tree is translated.

Consider a non-root node u of an input tree s. If an MTT translates s then, due to copying of input variables, there might be several states $q_1, \dots, q_n$ that translate u. Hence, it suffices to require that each x_i appears in at least one of the right-hand sides of the (q_ν, σ)-rules, $\nu \in [n]$, where σ is the label of u. Consider the following rules:

$$\begin{aligned}
\langle q_0, \gamma(x_1)\rangle \quad\;\; &\to b(\langle q_1, x_1\rangle, \langle q_2, x_1\rangle) \\
\langle q_1, \sigma(x_1, x_2)\rangle &\to \langle q, x_2\rangle \\
\langle q_2, \sigma(x_1, x_2)\rangle &\to \langle q, x_1\rangle
\end{aligned}$$

Then the last two rules are deleting in the statical sense, but if both of them are applied to a σ-labeled node, then together they are *not* deleting. Thus, in order to determine if a set of rules is nondeleting, we have to know which states of M translate a particular node u of the input tree s. Denote by $\mathrm{Sts}_M(s, u)$ the set of states of M which translate u (i.e., the set of states that appear in M's state sequence of s at u, see Definition 3.7 in [EM99]). To determine the *state set* $\mathrm{Sts}_M(s, u)$ *of* s *at* u we can use a top-down relabeling (i.e., a particular LT) as follows.

Definition 5. Let $M = (Q, \Sigma, \Delta, q_0, R)$ be a parameter nondeleting and non-erasing MTT. First, we define, for a subset $\overline{Q}$ of Q, the set of states that appear with x_i in the right-hand sides $\mathrm{rhs}_M(\overline{Q}, \sigma)$.

$$\mathrm{States}_M(\overline{Q}, \sigma, i) := \{q \mid \exists q' \in \overline{Q} \text{ such that } \langle q, x_i\rangle \text{ appears in } \mathrm{rhs}_M(q', \sigma)\}.$$

Define the linear top-down tree transducer $\mathrm{Sts}(M) = (\mathcal{P}(Q), \Sigma, \langle \Sigma, \mathcal{P}(Q)\rangle, \{q_0\}, R_S)$. For every $\overline{Q} \subseteq Q$ and $\sigma \in \Sigma^{(k)}$ let the rule

$$\langle \overline{Q}, \sigma(x_1, \dots, x_k)\rangle \to \langle \sigma, \overline{Q}\rangle(\langle Q_1, x_1\rangle, \dots, \langle Q_k, x_k\rangle)$$

be in R_S, where for each $i \in [k]$, $Q_i = \mathrm{States}_M(\overline{Q}, \sigma, i)$. □

Consider the linear top-down tree transducer $\mathrm{Sts}(M)$ from Definition 5. Since M is parameter nondeleting, it should be clear that

$$\text{for all } s \in T_\Sigma \text{ and } u \in V(s) : \tau_{\mathrm{Sts}(M)}(s)[u] = \langle s[u], \mathrm{Sts}_M(s, u)\rangle.$$

We can now define the notion of a nondeleting MTT.

Definition 6. An MTT $M = (Q, \Sigma, \Delta, q_0, R)$ is *nondeleting*, if for every $s \in T_\Sigma$ and $u \in V(s)$, $\mathrm{Sts}_M(s, u) \neq \varnothing$. □

In the next lemma it is proved that for every parameter nondeleting and nonerasing MTT M there are a linear top-down tree transducer N and an MTT M' such that the composition of N followed by M' equals M's translation, and M' is (additionally) nondeleting. The LT N is a variation of $\mathrm{Sts}(M)$ from Definition 5: N does not relabel nodes u with $\mathrm{Sts}_M(s, u) = \varnothing$, but deletes them.

Lemma 7. Let M be a parameter nondeleting and nonerasing MTT. There are an LT N and an MTT M' such that (1) $\tau_N \circ \tau_{M'} = \tau_M$ and (2) M' is nondeleting, parameter nondeleting, and nonerasing. If M is linear, then so is M'.

Proof. Let $M = (Q, \Sigma, \Delta, q_0, R)$ and let $\mathrm{Sts}(M) = (\mathcal{P}(Q), \Sigma, \langle \Sigma, \mathcal{P}(Q) \rangle, \{q_0\}, R_S)$ be as in Definition 5. Then, N is obtained from $\mathrm{Sts}(M)$ by removing the state $\varnothing$ from $\mathcal{P}(Q)$ and the corresponding rules from R_S, and erasing all $\langle \varnothing, x_i \rangle$ from the right-hand sides of the rules, changing the rank of an output symbol $\langle \sigma, \overline{Q} \rangle$ of rank k' into $\langle \sigma, \overline{Q} \rangle \in \Sigma_N^{(k)}$ with $k = |\{i \in [k'] \mid \mathrm{States}_M(\overline{Q}, \sigma, i) \neq \varnothing\}|$.

We now define the new MTT $M' = (Q, \Sigma_N, \Delta, q_0, R')$. For every $q \in Q^{(m)}$, $\langle \sigma, \overline{Q} \rangle \in \Sigma_N^{(k)}$, with $m, k \geq 0$, let the rule

$$\langle q, \langle \sigma, \overline{Q} \rangle (x_1, \ldots, x_k) \rangle (y_1, \ldots, y_m) \to \zeta$$

be in R'. If $q \notin \overline{Q}$ then let ζ be an arbitrary (dummy) tree, such that the rule is nondeleting, parameter nondeleting, and nonerasing. If $q \in \overline{Q}$, then let

$$\zeta = \mathrm{rhs}_M(q, \sigma)[\langle q', x_{i_j} \rangle \leftarrow \langle q', x_j \rangle \mid q' \in Q, j \in [k]],$$

where $x_{i_1}, \ldots, x_{i_k}$ with $i_1 < \cdots < i_k$ are all elements of X that appear in $\mathrm{rhs}_M(\overline{Q}, \sigma)$. It should be clear that M' is nondeleting, and that $\tau_N \circ \tau_{M'} = \tau_M$. In fact, M' and M have the same (nonempty) state sets on "corresponding" nodes (i.e., nodes that were not deleted by N). □

3.3 Removing the Skipping of Monadic Input Nodes

Another phenomenon which is harmful to the linear bounded input property is caused by the presence of monadic input symbols: they can be "skipped". A rule of an MTT is *skipping*, if it is of the form

$$\langle q, \gamma(x_1) \rangle (y_1, \ldots, y_m) \to \langle q', x_1 \rangle (y_{j_1}, \ldots, y_{j_m}) \qquad \text{"skipping" of } \gamma$$

where $j_1, \ldots, j_m$ is a permutation of the numbers $1, \ldots, m$. If all the states translating a monadic input node u have skipping rules, then their application does not change the size of the sentential form (while u is "consumed"). If an input tree can have arbitrarily many such nodes u then its size cannot be bounded. Hence, to guarantee linear bounded input, at least one rule that is applied to a monadic input node must be nonskipping.

Definition 8. An MTT $M = (Q, \Sigma, \Delta, q_0, R)$ is *nonskipping*, if for every $s \in T_\Sigma$ and $u \in V(s)$ with $\gamma = s[u] \in \Sigma^{(1)}$, there exists a $q \in \mathrm{Sts}_M(s, u)$ such that the (q, γ)-rule is nonskipping. □

If M is a nondeleting and parameter nondeleting MTT, then a dynamical consequence is that every node of an input tree s is translated by at least one state of M. If M is additionally nonerasing and nonskipping, then we call it productive. A dynamical consequence of the fact that M is productive, is that it is of linear bounded input, which will be proved in Lemma 11 for the cases that M is linear or a top-down tree transducer.

Definition 9. An MTT is *productive*, if it is parameter nondeleting, nonerasing, nondeleting, and nonskipping. The corresponding class of translations is denoted by $\mathrm{MTT}_{\mathrm{prod}}$ (and by $\mathrm{LMTT}_{\mathrm{prod}}$ or $\mathrm{T}_{\mathrm{prod}}$ if the MTTs are linear or top-down tree transducers, respectively). □

As the reader may verify, the MTT of Example 2 is parameter nondeleting and nondeleting, but not productive, because it is erasing and skipping; note also that it is linear. In Lemma 10 it is proved that for every parameter nondeleting, nonerasing, and nondeleting MTT M there are a linear top-down tree transducer N and an MTT M' such that the composition of N followed by M' equals M's translation, and M' is productive. As in the proof of Lemma 7, the LT N is a variation of $\mathrm{Sts}(M)$ from Definition 5: N does not relabel nodes u for which all the $(q, s[u])$-rules, $q \in \mathrm{Sts}_M(s, u)$, are skipping, but deletes them.

Lemma 10. Let M be parameter nondeleting and nonerasing. There is an LT N and an MTT M' such that $\tau_N \circ \tau_{M'} = \tau_M$ and M' is productive. If M is linear, then so is M'. If M is a top-down tree transducer, then so is M'.

Proof. By Lemma 7 there are an LT N_1 and an MTT M_1 such that $\tau_{N_1} \circ \tau_{M_1} = \tau_M$, and M_1 is nondeleting, parameter nondeleting, and nonerasing (and if M is linear, then so is M_1). We now construct the linear top-down tree transducer N_2 and the productive MTT M' such that $\tau_{N_2} \circ \tau_{M'} = \tau_{M_1}$. Since LT is closed under composition (see, e.g., Corollary 3.41 of [FV98]), there is an LT N such that $\tau_N = \tau_{N_1} \circ \tau_{N_2}$. Since the construction of N_2 and M' is similar to the construction given in the proof of Lemma 7, we just give a sketch.

Let $M_1 = (Q, \Sigma, \Delta, q_0, R)$. Again N_2 is very similar to $\mathrm{Sts}(M_1)$: if it translates a node u of an input tree s, then it has in its state the nonempty subset $\mathrm{Sts}_{M_1}(s, u)$ of states of M_1 that translate u. Moreover, it keeps a mapping d in its states, that associates with a state $q \in Q^{(m)}$ a pair (r, π), where $r \in Q^{(m)}$ and π is a permutation of the numbers in $[m]$. If N_2 translates, with $\overline{Q}$ and d in its state, a monadic node u of the input tree labeled γ for which all (r, γ)-rules with $r \in \overline{Q}$ are skipping (a "skipping γ"), then it deletes this node, and keeps in its state the new mapping d' that changes each (r, π) into (r', π'), where r' is the state that appears in $\mathrm{rhs}_{M_1}(r, \gamma) = \zeta$ and π' is the composition of π with the permutation of the parameters in ζ. If N_2 translates in state $(\overline{Q}, d)$ a nonskipping symbol σ then it relabels it by $\langle \sigma, d \rangle$.

The MTT M' has Q as set of states. For a state $q \in Q$ and an input symbol $\langle \sigma, d\rangle$ the right-hand side of the $(q, \langle \sigma, d\rangle)$-rule is obtained from $\mathrm{rhs}_{M_1}(r, \sigma)$ by replacing each y_j by $y_{\pi^{-1}(j)}$, where $d(q) = (r, \pi)$. □

Next, it is shown that productive MTTs have linear bounded input. In fact, even though this can be proved for arbitrary MTT, we only treat linear MTTs and top-down tree transducers (which both can be realized by attribute grammars).

Lemma 11. $(\mathrm{LMTT}_{\mathrm{prod}} \cup \mathrm{T}_{\mathrm{prod}}) \subseteq \mathrm{LBI}$.

Proof. Let M be an $\mathrm{LMTT}_{\mathrm{prod}}$ or a $\mathrm{T}_{\mathrm{prod}}$. First, neglect the fact that M is nonskipping. Then, the size of an output tree $t = \tau_M(s)$ is greater than or equal to the number of leaves of s. This is because each leaf of s is translated by M and generates at least one output symbol, due to the fact that M is nondeleting, parameter nondeleting, and nonerasing. This also implies that the number $\#_{\geq 2}(s)$ of symbols of rank greater than one in s is linear bounded by $|t|$, because for any tree s, $\#_{\geq 2}(s) < \mathrm{leaves}(s)$, where $\mathrm{leaves}(s)$ is the number of leaves in s.

Thus, it remains to show that the number $\mathrm{mon}(s)$ of monadic nodes of s is bounded by $|t|$. Since M is nonerasing, nondeleting, parameter nondeleting, and linear or a top-down tree transducer, a nonskipping rule $l \to r$ generates at least one output symbol α: either r contains α or it contains at least two occurrences of a parameter y such that the actual parameter tree for y contains α. Therefore, $\mathrm{mon}(s) < |t|$. To be more precise, we obtain $\mathrm{leaves}(s) + \mathrm{mon}(s) \leq |t|$, because each leaf *and* each monadic symbol generates at least one symbol of t. Altogether,

$$\begin{aligned}|s| &= \mathrm{leaves}(s) + \mathrm{mon}(s) + \#_{\geq 2}(s)\\ &< \mathrm{leaves}(s) + \mathrm{mon}(s) + \mathrm{leaves}(s)\\ &\leq 2(\mathrm{leaves}(s) + \mathrm{mon}(s))\\ &\leq 2|t|.\end{aligned}$$

This shows that τ_M has linear bounded input by constant $c = 2$. □

We are now ready to state the main theorem: A composition τ of MTTs can be realized by $\tau' \circ \tau_{\mathrm{prod}}$, where τ_{prod} is a composition of productive transducers and τ' is the composition of a relabeling and a linear top-down tree transducer.

Theorem 12. For $k \geq 1$, $\mathrm{MTT}^k \subseteq \mathrm{D_tQRELAB} \circ \mathrm{LT} \circ \mathrm{T}_{\mathrm{prod}} \circ \mathrm{LMTT}^k_{\mathrm{prod}}$.

Proof. Let $\tau \in \mathrm{MTT}^k$ with $k \geq 1$. By Theorems 4.6 and 4.8 of [EV85] τ is in $\mathrm{MTT}^{k-1} \circ \mathrm{T} \circ \mathrm{LMTT}$, which by Lemmas 4 and 10 and the fact that the class $\mathrm{D_tQRELAB} \circ \mathrm{T}$ is closed under composition (see, e.g., Theorem 3.39 of [FV98] and Theorem IV.4.11(ii) of [GS84]) is included in $\mathrm{MTT}^{k-1} \circ \mathrm{D_tQRELAB} \circ \mathrm{T} \circ \mathrm{LMTT}_{\mathrm{prod}}$. If $k \geq 2$ then, since the class MTT is closed under right composition with both $\mathrm{D_tQRELAB}$ and T (by Lemma 11 of [EM02] and Theorem 4.12 of [EV85], respectively), the above is included in $\mathrm{MTT}^{k-1} \circ \mathrm{LMTT}_{\mathrm{prod}}$. Hence, by induction $\mathrm{MTT}^k \subseteq \mathrm{D_tQRELAB} \circ \mathrm{T} \circ \mathrm{LMTT}^k_{\mathrm{prod}}$. Since every top-down tree transducer is a parameter nondeleting and nonerasing MTT we obtain, by Lemma 10, that $\mathrm{T} \subseteq \mathrm{LT} \circ \mathrm{T}_{\mathrm{prod}}$, which with the above proves the lemma. □

Since relabelings in $\mathrm{D_tQRELAB}$ and linear top-down tree transducers preserve regular tree languages (Lemma IV.6.5 and Corollary IV.6.6 of [GS84], respectively), Theorem 12 gives the following corollary.

Corollary 13. $\mathrm{MTT}^*(\mathrm{REGT}) = \mathrm{LMTT}^*_{\mathrm{prod}}(\mathrm{T}_{\mathrm{prod}}(\mathrm{REGT}))$.

3.4 Deciding the Finiteness of Ranges

The previous results of this section can be used to prove that, for a tree language L in $\mathrm{MTT}^*(\mathrm{REGT})$ it is decidable whether or not L is finite. This result was proved in [DE98], even for (compositions of) nondeterministic MTTs.

Theorem 14. The finiteness of languages in $\mathrm{MTT}^*(\mathrm{REGT})$ is decidable.

Proof. Let L be in $\mathrm{MTT}^*(\mathrm{REGT})$. By Corollary 13, Lemma 11, and the fact that LBI is closed under composition, $L = \tau(R)$ with $\tau \in \mathrm{LBI}$ and $R \in \mathrm{REGT}$. Since τ has linear bounded input by some c, $|s| \leq c|\tau(s)|$ for all $s \in R$. Hence, if L is finite, then R is finite, because all trees in R are of size bounded by $c|t|$ where t is the largest tree in L. Thus, L is finite if and only if R is finite. Since finiteness of regular tree languages is decidable, this proves the lemma. □

4 Complexity Results

Our first complexity result says that, for a composition τ of MTTs, there is a Random Access Machine (RAM) that computes, for every input s, the output tree $t = \tau(s)$ in time linear in $|s| + |t|$.

Theorem 15. For every $\tau \in \mathrm{MTT}^*$ there is a RAM that computes, given an input tree s, the output $t = \tau(s)$ in time $O(|s| + |t|)$.

Proof. Let $\tau \in \mathrm{MTT}^k$, $k \geq 1$. By Theorem 12, $\tau \in \mathrm{D_tQRELAB} \circ \mathrm{LT} \circ \mathrm{T}_{\mathrm{prod}} \circ \mathrm{LMTT}^k_{\mathrm{prod}}$. It should be clear how to realize a relabeling in time linear in $|s|$. Top-down tree transducers are (particular) attribute grammars (AGs) (see, e.g., Chapter 6 of [FV98]), and each linear MTT M can be simulated by the composition of a total deterministic top-down relabeling followed by an AG (by the proof of Lemma 5.9 and Lemma 5.12 of [EM99], and because M is "single use restricted in the input" (suri), cf. Definition 5.7 of [EM99]). Therefore, τ can be computed by the composition of $k+2$ AGs (and $k+1$ relabelings). As discussed in the Introduction, the tree to tree translation of an attribute grammar A can be computed in time linear in the sum of the sizes of the input and output trees (on a RAM R_A). Since all but the first translation (of the linear top-down tree transducer) have linear bounded input, the size of each intermediate result is linear bounded by $|\tau(s)|$. Hence, each of the RAMs R_A runs in time linear in $|s| + |t|$, which concludes the proof. □

Note that for a pebble tree transducer τ (cf. the Introduction) it is known that $t = \tau(s)$ can be computed in time polynomial in $|s| + |t|$ (cf. Proposition 3.5 of [MSV00]); hence, together with the result of [EM] that pebble tree transducers can be simulated by compositions of MTTs, the above theorem is an improvement of their result.

By Theorem 2.5 of [Pap94], it follows from Theorem 15 that for every $\tau \in \text{MTT}^*$ there is a deterministic Turing Machine M_τ that, given s, computes the output tree $t = \tau(s)$ in time $O(m^3)$, where $m = |s| + |t|$. We now turn to our second complexity result, about the output languages in $\text{MTT}^*(\text{REGT})$.

Theorem 16. $\text{MTT}^*(\text{REGT}) \subseteq \text{DSPACE}(n)$.

Proof. (sketch) Let $M = (Q, \Sigma, \Delta, q_0, R)$ be a T or an LMTT and let $\#_M$ be a new binary symbol. We now construct the transducer M_1 which has as output language $\{\#_M(s, \tau_M(s)) \mid s \in T_\Sigma\}$. The transducer M_1 is obtained from M by adding the new state id of rank zero, and for every $q \in Q$ the new state q_{id} of rank zero. These new states are used to copy the input tree. For every $\sigma \in \Sigma^{(k)}$, M_1 has the rules $\langle \text{id}, \sigma(x_1, \dots, x_k)\rangle \to \sigma(\langle \text{id}, x_1\rangle, \dots, \langle \text{id}, x_k\rangle)$ and $\langle q_0, \sigma(x_1, \dots, x_k)\rangle \to \#_M(\sigma(\langle r_1, x_1\rangle, \dots, \langle r_k, x_k\rangle), \zeta)$ and $\langle q_{\text{id}}, \sigma(x_1, \dots, x_k)\rangle \to \sigma(\langle p_1, x_1\rangle, \dots, \langle p_k, x_k\rangle)$, where $\zeta = \text{rhs}_M(q_0, \sigma)$, $r_i = q_{\text{id}}$ and q is chosen such that $\langle q, x_i\rangle$ occurs in ζ, and $r_i = \text{id}$ if no element of $\langle Q, x_i\rangle$ occurs in ζ (and similarly for p_i and $\text{rhs}_M(q, \sigma)$). Since M_1 is defined in such a way that it is either a top-down tree transducer, or an MTT that is suri (cf. Definition 5.7 of [EM99]), it can, as outlined in the proof of Theorem 15, be realized by an attribute grammar A. It follows from [Eng86] that there is a TM T_A in $\text{DSPACE}(\log^2 n)$ which recognizes the output language of A.

Now let L be a language in $\text{MTT}^*(\text{REGT})$. By Corollary 13, it can be computed by a composition of productive Ts and LMTTs, for each one of which there is an AG that computes its translation as output language. By Lemma 11, the size of all intermediate results are linear bounded by the size of the given tree t. Hence, a TM T_L can be constructed that enumerates all possible intermediate results and verifies them (using the T_As of above) in such a way that T_L operates in $\text{DSPACE}(n)$. □

Note that, since MTTs can realize yield mappings (see Example 2), Theorem 16 implies that also the *MTT hierarchy* of string output languages in $y\text{MTT}^*(\text{REGT})$ is in $\text{DSPACE}(n)$. Since the IO-hierarchy is included in the MTT hierarchy, we obtain the following corollary. Note that it was proved in [Fis68] already that the IO languages (= level one of the IO-hierarchy) are in $\text{NSPACE}(n)$; in [Asv81] this result was improved to LOG(CFL). From Corollary 8.12 of [Dam82] it is known that languages in the IO-hierarchy are recursive.

Corollary 17. The IO hierarchy is in $\text{DSPACE}(n)$.

Note that by Theorem 36 of [EM02] the EDT0L control hierarchy is included in the IO hierarchy.

Acknowledgements

I thank Joost Engelfriet for inspiring me to work on this topic, Dirk Nowotka for initial discussions, and Janis Voigtländer and the referees for helpful comments.

References

[Asv81] P.R.J. Asveld. Time and space complexity of inside-out macro languages. *Internat. J. Comput. Math.*, 10:3–14, 1981.

[Bak78] B. S. Baker. Generalized syntax directed translation, tree transducers, and linear space. *SIAM J. Comput.*, 7(3):376–391, 1978.

[CF82] B. Courcelle and P. Franchi-Zannettacci. Attribute grammars and recursive program schemes. *TCS*, 17:163–191 and 235–257, 1982.

[Dam82] W. Damm. The IO- and OI-hierarchies. *TCS*, 20:95–207, 1982.

[DE98] F. Drewes and J. Engelfriet. Decidability of finiteness of ranges of tree transductions. *Inform. and Comput.*, 145:1–50, 1998.

[DJL88] P. Deransart, M. Jourdan, and B. Lorho. *Attribute Grammars, Definitions and Bibliography.* Number 323 in LNCS. Springer-Verlag, 1988.

[EM] J. Engelfriet and S. Maneth. Work in progress.

[EM99] J. Engelfriet and S. Maneth. Macro tree transducers, attribute grammars, and MSO definable tree translations. *Inform. and Comput.*, 154:34–91, 1999.

[EM02] J. Engelfriet and S. Maneth. Output string languages of compositions of deterministic macro tree transducers. *JCSS*, 64(2):350–395, 2002.

[Eng75] J. Engelfriet. Bottom-up and top-down tree transformations — a comparison. *Math. Systems Theory*, 9(3):198–231, 1975.

[Eng86] J. Engelfriet. The complexity of languages generated by attribute grammars. *SIAM J. Comput.*, 15(1):70–86, 1986.

[EV85] J. Engelfriet and H. Vogler. Macro tree transducers. *JCSS*, 31:71–146, 1985.

[Fis68] M.J. Fischer. *Grammars with macro-like productions.* PhD thesis, Harvard University, Massachusetts, 1968.

[Fül81] Z. Fülöp. On attributed tree transducers. *Acta Cybernetica*, 5:261–279, 1981.

[FV98] Z. Fülöp and H. Vogler. *Syntax-Directed Semantics – Formal Models based on Tree Transducers.* EATCS Monographs on Theoretical Computer Science (W. Brauer, G. Rozenberg, A. Salomaa, eds.). Springer-Verlag, 1998.

[GS84] F. Gécseg and M. Steinby. *Tree Automata.* Akadémiai Kiadó, Budapest, 1984.

[Knu68] D.E. Knuth. Semantics of context-free languages. *Math. Systems Theory*, 2:127–145, 1968. (Corrections in *Math. Systems Theory*, 5:95-96, 1971).

[Küh98] A. Kühnemann. Benefits of tree transducers for optimizing functional programs. In V. Arvind and R. Ramanujam, editors, *Proc. FST&TCS'98*, volume 1530 of *LNCS*, pages 146–157. Springer-Verlag, 1998.

[MSV00] T. Milo, D. Suciu, and V. Vianu. Typechecking for XML transformers. In *Proc. PODS'2000*, pages 11–22. ACM Press, 2000.

[Pap94] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, 1994.

[Via01] V. Vianu. A Web Odyssey: From Codd to XML. In *Proc. PODS'2001*, pages 1–15. ACM Press, 2001.

[Vog91] H. Vogler. Functional description of the contextual analysis in block-structured programming languages: a case study of tree transducers. *Science of Computer Programming*, 16(3):251–275, 1991.

[Wad90] P. Wadler. Deforestation: Transforming programs to eliminate trees. *TCS*, 73:231–248, 1990.

On the Hardness of Approximating Minimum Monopoly Problems

S. Mishra[1], Jaikumar Radhakrishnan[2], and S. Sivasubramanian[2]

[1] The Institute of Mathematical Sciences, Chennai 600113, India
soun@imsc.ernet.in
[2] School of Technology and Computer Science
Tata Institute of Fundamental Research, Mumbai 400005, India
{jaikumar,skrishna}@tcs.tifr.res.in

Abstract. We consider inapproximability for two graph optimisation problems called monopoly and partial monopoly. We prove that these problems cannot be approximated within a factor of $(\frac{1}{3} - \epsilon) \ln n$ and $(\frac{1}{2} - \epsilon) \ln n$, unless $\mathsf{NP} \subseteq \mathsf{Dtime}(n^{O(\log \log n)})$, respectively. We also show that, if Δ is the maximum degree in a graph G, then both problems cannot be approximated within a factor of $\ln \Delta - O(\ln \ln \Delta)$, unless $\mathsf{P} = \mathsf{NP}$, though both these problems can be approximated within a factor of $ln(\Delta) + O(1)$. Finally, for cubic graphs, we give a 1.6154 approximation algorithm for the monopoly problem and a $\frac{5}{3}$ approximation algorithm for partial monopoly problem, and show that they are APX-complete.

1 Introduction

1.1 The Problems

We consider the minimum monopoly problem and the minimum partial monopoly problem on graphs. These problems are defined as follows.

Monopoly: Given an undirected simple graph $G = (V, E)$, find the smallest possible set $X \subseteq V$ such that for all $v \in V$, $|X \cap N[v]| \geq \frac{1}{2}|N[v]|$, where $N[v]$, the *closed neighbourhood of* v, is the set of vertices at distance at most 1 from v.

Partial Monopoly: Given an undirected simple graph $G = (V, E)$, find the smallest possible set $X \subseteq V$ such that for all $v \in V - X$, $|X \cap N[v]| \geq \frac{1}{2}|N[v]|$.

These problems arise in the study of *local majority processes* [11]. Here, there is a network of processors, where each processor is initially assigned a colour, either red or green. In each step, each processor uses a voting system to replace its colour by the colour that appears most often in its closed neighbourhood. Several rules for resolving ties have been considered in the literature [4]. The rule we use is the following: prefer the colour *red* when there is a tie. Here processors coloured red are to be thought of as faulty, or deviant in some way, and the green processor as faultless. Voting systems are widely used to ensure integrity in fault tolerant systems [9,2]. In the distributed setting, majority voting has been used as a tool for fault mending using local information [8]. In the abstraction we consider, a processor is assumed to start malfunctioning if a majority of processors around it are faulty. The rule for resolving ties that we adopt corresponds to the following pessimistic assumption: the processor does malfunction even if exactly half the processors around it are faulty.

In the literature, one also considers a situation where a coalition of faulty processors tries to take *control* of the system. Since the processors malfunction with malicious intent, they themselves are not expected to perform any self-correcting routine. Thus, in order for them to take control of the entire system, it is enough if they can influence all processors outside the coalition. When translated into the language of graphs, this corresponds to the partial monopoly problem described above. For a survey of results and problem in this area see *Local Majority Voting, Small Coalitions and Controlling Monopolies in Graphs: A Review* by D. Peleg [11].

1.2 Previous Results and Conjectures

These problems are NP-complete [11]. Peleg [11] considers approximation algorithms for these problems and observes that a variant of the greedy set-cover algorithm gives a factor $\ln \mathit{\Delta} + O(1)$, where $\mathit{\Delta}$ is the maximum degree in G. He also conjectures, based on the hardness results of Lund and Yannakakis [10], and Feige [3], for the set-cover problem, that for any $\epsilon > 0$ the monopoly problem has no $\ln n - \epsilon$ approximation algorithm, unless $\mathsf{NP} \subseteq \mathsf{Dtime}(n^{O(\log\log n)})$, where n is the number of vertices in G [1].

1.3 Our Results

Inapproximability: We prove a weak form of Peleg's conjecture. We prove,

Theorem 1. *(a) For all $\epsilon > 0$, there is no approximation algorithm for the monopoly problem with approximation ratio $(\frac{1}{3} - \epsilon)\ln n$, unless $\mathsf{NP} \subseteq \mathsf{Dtime}(n^{O(\log\log n)})$.*
(b) For all $\epsilon > 0$, there is no approximation algorithm for the partial monopoly problem with approximation ratio $(\frac{1}{2} - \epsilon)\ln n$, unless $\mathsf{NP} \subseteq \mathsf{Dtime}(n^{O(\log\log n)})$.

Similar statements hold for a variant of the partial monopoly problem, where a processor consults all other processors at most distance two away. For want of space, we omit a proof of this result.

For the set cover problem, Trevisan [12] considers inapproximability results when the sets are of size at most B, and shows that if there is an approximation algorithm with factor at most $\ln B - O(\ln\ln B)$, then $\mathsf{P} = \mathsf{NP}$. We show a similar result for our problems.

[1] We believe that instead of $\ln n - \epsilon$ actually $(1 - \epsilon)\ln n$ was meant.

Theorem 2. *(a) For every ϵ there exists absolute constants, Δ_ϵ and C such that for any $\Delta \geq \Delta_\epsilon$, any partial monopoly instance with degree bounded by Δ cannot be approximated to within a factor $\ln \Delta - C \ln\ln \Delta$, unless $\mathsf{P} = \mathsf{NP}$.*
(b) For every ϵ there exists absolute constants, Δ_ϵ and C such that for any $\Delta \geq \Delta_\epsilon$, any monopoly instance with degree bounded by Δ cannot be approximated to within a factor $\ln \Delta - C \ln\ln \Delta$, unless $\mathsf{P} = \mathsf{NP}$.

Regular Graphs: It is also known that variants of NP-hard graph optimisation problems in which the degree of the graph is bounded by a constant often allows one to achieve better approximation. For example, minimum dominating set for an arbitrary graph is not approximable within a constant factor [3], whereas it is known to be constant factor approximable for 3-regular graphs [1]. For a list of NP-hard problems having similar properties we refer the reader to [5,1,13]. Therefore, it would be desirable to identify how much "bounding" the degree in a graph is helpful in obtaining solutions, both exact and approximate.

For these reasons, we consider the case of regular graphs separately. In fact, the size of the minimum monopoly and the minimum partial monopoly in regular graphs is a constant factor of the number of vertices: $\frac{1}{2}|V(G)|$ for monopoly and $\frac{1}{3}|V(G)|$ for partial monopoly. Thus, there exist trivial factor 2 and 3 approximation (namely, $|V(G)|$) for these problems. We first improve on these trivial algorithms for 3-regular graphs.

Theorem 3. *(a) The monopoly problem on 3-regular graphs can be approximated to a factor 1.6154 in polynomial time.*
(b) The partial monopoly problem on 3-regular graphs can be approximated to a factor $\frac{5}{3}$ in polynomial time.

We also ask if these bounds can be improved and moved arbitrarily close to 1. This is not possible, even for 3-regular graphs, unless $\mathsf{P} = \mathsf{NP}$.

Theorem 4. *(a) The monopoly problem for 3-regular graphs is APX-complete.*
(b) The partial monopoly problem for 3-regular graphs is APX-complete.

2 The Set-Cover Problem

Our results for the general monopoly problems are based on the results for the set-cover problem.

Definition 1 (Set-cover). *Given a set-system $\mathcal{C} = (\mathcal{U}, \mathcal{F})$, where $\mathcal{U}$ is the universe and $\mathcal{F}$ is a collection of subsets of $\mathcal{U}$ such that $\bigcup_{F \in \mathcal{F}} F = \mathcal{U}$, find a minimum size subcollection $\mathcal{F}' \subseteq \mathcal{F}$, so that $\bigcup_{F \in \mathcal{F}'} F = \mathcal{U}$. Let $\text{OptSC}(\mathcal{C})$ denote the size of the smallest such subcollection. Let $\Delta(\mathcal{C})$ be the size of the largest set in $\mathcal{F}$ and $\deg(\mathcal{C})$ be the maximum number of sets containing an element of $\mathcal{U}$. In a variant of this problem, known as the* set multicover problem *we are given requirements $\text{req}(u)$ for each element u of the universe, such that u appears in at least $\text{req}(u)$ sets in $\mathcal{F}'$. We are then required to find a subcollection $\mathcal{F}'$ so that u appears in at least $\text{req}(u)$ sets in $\mathcal{F}'$.*

The decision version of the set-cover problem ('Is $\textsc{OptSC}(C) \leq k$?') is NP-complete. It is well-known [15] that the *greedy* heuristic produces a solution that is always within a factor $1+\ln \Delta(\mathcal{C})$ of the optimum. This holds for the set-cover problem as well as the set multicover problem. For the set-cover problem, Slavík has shown that the greedy solution is never more than factor $\ln \Delta - \ln\ln \Delta + 0.78$ bigger than the optimum.

Strong inapproximability results are known for this problem.

Theorem 5 (Feige [3]). *If polynomial-time algorithm that approximates the optimum set-cover within a factor of* $(1-\epsilon)\ln n$ *for some* $\epsilon > 0$ *(where* $n = |\mathcal{U}| + |\mathcal{F}|$*), then* $\mathsf{NP} \subseteq \mathsf{Dtime}(n^{O(\log\log n)})$.

This theorem is obtained from results on probabilistically checkable proofs. In fact, Feige shows that, for each $\epsilon > 0$, an instance φ of SAT (say) of size n, can be converted to an instance $\mathcal{C}$ of set-cover of size $N = n^{\theta((\log\log n)/\epsilon)}$ such that if the φ is satisfiable, then $\mathcal{C}$ has a cover of size k (easily computable from n), but if φ is not satisfiable, then $\mathcal{C}$ has no set-cover of size $(1-\epsilon)\ln N$. While deriving the hardness results for the monopoly problem and the partial monopoly problem, we will exploit features of Feige's reduction. The instance $\mathcal{C} = (\mathcal{U}, \mathcal{F})$ has the following features. There is a parameter ℓ whose value is fixed at $\theta(\log\log n)$ and another parameter m, whose value is fixed at $n^{\theta(\ell/\epsilon)}$. Then,

$$|\mathcal{U}| = mn^{\ell}\text{polylog}n; \quad |\mathcal{F}| = n^{\ell}\text{polylog}n;$$

$$\Delta(\mathcal{C}) \leq m\text{polylog}n; \quad \deg(\mathcal{C}) \leq \text{polylog}n.$$

Trevisan [12] analysed Feige's construction in order to prove inapproximability results depending on the size of the largest set in the set-system. Let Δ set-cover problem be the set-cover problem where all sets have size at most Δ.

Theorem 6. *There is an absolute constant* C *and* Δ_0*, such that for all* $\Delta \geq \Delta_0$*, if there is a polynomial-time algorithm for the* Δ *set-cover problem that approximates the optimum within a factor of factor* $\ln \Delta - C \ln\ln \Delta$*, then* $\mathsf{P} = \mathsf{NP}$.

Again, while using this theorem we will exploit some features of the reduction from an instance φ of SAT to an instance of Δ set-cover. There are parameters ℓ and m as before; this time, ℓ is fixed to $\theta(\ln\ln \Delta)$ and m is fixed to $\Delta/\text{polylog}\Delta$. Then,

$$|\mathcal{U}| = mn^{\ell}\text{polylog}\Delta; \quad |\mathcal{F}| = n^{\ell}\text{polylog}\Delta;$$

$$\Delta(\mathcal{C}) \leq \Delta; \quad \deg(\mathcal{C}) \leq \text{polylog}\Delta.$$

3 Hardness Results

We will first prove the results we have for the partial monopoly problem and then modify certain constructions appropriately to prove similar results for the monopoly problem.

3.1 Proof of Theorem 1(b)

We will show that given an instance of the set cover problem, we can efficiently transform it to an instance of the partial monopoly problem, so that we can recover a solution for the set cover instance from a solution of the partial monopoly instance, with roughly the same approximation ratio.

Fix $\epsilon > 0$. Let $\mathcal{C} = (\mathcal{U}, \mathcal{F})$ be an instance of the set cover problem, where $\mathcal{U}$ is the universe and $\mathcal{F}$ is the family of subsets of $\mathcal{U}$. Assume that $|\mathcal{U}| + |\mathcal{F}| = n$. For $u \in U$, let $\deg(u) = \{F \in \mathcal{F} : u \in F\}$.

Associated with $\mathcal{C}$ is a natural bipartite graph where the sets in $\mathcal{F}$ and the elements of $\mathcal{U}$ appear as vertices. An element is connected by an edge to all sets it belongs to. Let G_c denote this graph. To obtain our instance of the partial monopoly problem, we take $L = n$ disjoint copies of G_c. To these we add a set W of n new vertices that are connected to the rest as follows. Each vertex of G_c corresponding to the set $S \in \mathcal{F}$, is connected to $|S| + 1$ distinct vertices in W (note that $|W| = n \geq |S|+1$) ; each vertex corresponding to the element $u \in \mathcal{U}$ is connected to $\deg(u) - 1$ distinct vertices in W(note that $|W| = n \geq deg(u) - 1$). Let this graph with $n(L + 1)$ vertices be G.

Claim. $n + L \cdot \textsc{Opt}\mathrm{SC}(\mathcal{C}) \geq \textsc{Opt}\mathrm{PM}(G) \geq L \cdot \textsc{Opt}\mathrm{SC}(\mathcal{C})$.

Proof of claim: To see the first inequality, let I be a cover in $\mathcal{C}$. Then, W along with all copies of vertices corresponding to sets in I is a partial monopoly in G. To see the second inequality, let X be a partial monopoly in G. Consider the subset X_i of X corresponding to vertices in the ith copy of G_c. If any of these vertices corresponds to an element $u \in U$, exchange it for a vertex (from the same copy of G_c corresponding to a set $S \in \mathcal{F}$ containing u. Then, the resulting set X_i' corresponds to a set cover of $\mathcal{C}$. Thus, $|X_i| \geq \textsc{Opt}\mathrm{SC}(\mathcal{C})$, and $|X| \geq L \cdot \textsc{Opt}\mathrm{SC}(\mathcal{C})$. This, also shows how from a solution X to the partial monopoly problem we can recover a solution of size at most $(1/L)|X|$ for the set cover problem. (*end of claim*)

Now, suppose there is a polynomial time algorithm for approximating the partial monopoly problem within a factor of $(\frac{1}{2} - \epsilon) \ln N$, for graphs with N vertices. Then, from the above claim, we obtain in polynomial time an approximation for the set cover instance of size at most $(1/L)\textsc{Opt}\mathrm{PM}(G)(\frac{1}{2} - \epsilon) \ln(n(L+1))$. Now, by the first inequality in the above claim, this is at most

$$(\frac{n}{L} + \textsc{Opt}\mathrm{SC}(\mathcal{C}))(\frac{1}{2} - \epsilon) \ln(n(L + 1)) \leq (\frac{1}{2} - \epsilon) \ln(n^2 + n) \cdot (\textsc{Opt}\mathrm{SC}(\mathcal{C}) + 1)$$

We note that

$$(\frac{1}{2} - \epsilon) \ln(2n^2)(\textsc{Opt}(\mathcal{C}) + 1) \leq (\frac{1}{2} - \epsilon) \ln n^2 (1 + \frac{\ln 2}{\ln n^2})(\textsc{Opt}(\mathcal{C}) + 1)$$

As n increases, we note that the size of the optimal cover will also increase, and for sufficiently large n, we can bound the above by

$$(1 - 2\epsilon) \ln n (1 + \frac{\epsilon}{10})\textsc{Opt}(\mathcal{C})(1 + \frac{\epsilon}{10}) \leq (1 - \epsilon) \ln n \textsc{Opt}(\mathcal{C})$$

for large enough n. By Feige's theorem, this is impossible unless $NP \subseteq \mathsf{Dtime}(n^{O(\log\log n)})$. Hence, there is no factor $(\frac{1}{2} - \epsilon)\ln|V|$ approximation algorithm for the partial monopoly problem, unless $\mathsf{NP} \subseteq \mathsf{Dtime}(n^{O(\log\log n)})$.

3.2 Proof of Theorem 1(a)

We need to show how an instance of the set cover problem can be transformed to an instance of the monopoly problem, so that we can recover a solution for the set cover instance from a solution of the monopoly instance, with roughly the same approximation ratio.

Fix $\epsilon > 0$. Let $\mathcal{C} = (\mathcal{U}, \mathcal{F})$ be an instance of the set cover problem. As before, let G_c be the bipartite graph associated with C. To obtain our instance of the monopoly problem, we take $L \overset{\Delta}{=} n^2$ disjoint copies of G_c. To these we add a set W of n^2 new vertices, connected as a clique among themselves, and connected to the rest as follows. Each vertex of G_c corresponding to the set $S \in \mathcal{F}$, is connected to $|S| + 1$ vertices in W; each vertex corresponding to the element $u \in \mathcal{U}$ is connected to $\deg(u) - 1$ vertices in W. Thus, there are a total of at most $L \cdot n^2 \le n^4$ edges connecting between the copies of G_c and W. We allocate these edges to vertices of W as equally as possible; this ensures, in particular, that the degree of a vertex in W is at most $2L - 1 = 2n^2 - 1$. Let this graph with $L \cdot n + n^2$ vertices be G. Then, we have the following claim.

Claim. $n^2 + L \cdot \textsc{OptSC}(\mathcal{C}) \ge \textsc{OptPM}(G) \ge L \cdot \textsc{OptSC}(\mathcal{C})$.

The proof of this claim is identical to the proof of the corresponding claim for partial monopoly. In particular, one can recover from a solution X for the monopoly problem, a solution for the set cover problem of size at most $|X|/L$. The rest of the argument is similar. We omit it for want of space.

4 Graphs with Bounded Degree

4.1 Proof of Theorem 2(b)

Note that the maximum size of a set in these hard set-cover instances is large when compared to the number of sets which contain any point. Thus, in the natural bipartite graph, the degree of sets will be larger than the degree of a point. Theorem 6 says that for all $\epsilon > 0$, there exists a Δ_ϵ such that for any $\Delta > \Delta_\epsilon$ unless $\mathsf{NP} = \mathsf{P}$, there exist instances of the set-cover problem with each set having at most Δ points such that no polynomial time algorithm can approximate these instances to a factor $\ln\Delta - O(\ln\ln\Delta)$. Here Δ depends on ϵ and there are hard instances for infinitely many n.

Since our graphs have degree bounded by Δ, it follows that $m5^{(k-1)l}3^l \le \Delta$. This imposes a value on m to be $\Delta/polylog(\Delta)$.

We embed a hard to approximate instance of set-cover to obtain an instance for the partial monopoly as before but this time without taking multiple copies. This happens because the size of the set-cover instance is only a function of n

(and Δ) in this case, whereas in the general case, it is a function of both m and n. We just need to check that the number of vertices in W is less than the optimal value of the set-cover and that these new vertices have degree less than $f(\Delta)$ for some function f.

Adding edges as before, one can check that we need to add $(5n/3)^l 7^l \Delta$ edges to the set W and if we have $(5n/3)^l/\epsilon$ new vertices to which we add edges equitably, the degree of a vertex in W will be $\epsilon \Delta 7^l$ which is $\epsilon \Delta polylog(\Delta)$. Thus we can use Theorem 6 to conclude our theorem.

4.2 Proof of Theorem 2(a)

The idea is identical to that applied for proving hardness for the monopoly problem. The only difference is that we do not add a clique on the set W as W has $(5n/3)^l/\epsilon$ vertices and we want the degree of a vertex in W to be less than $f(\Delta)$ for some function f.

Since Δ is a constant and n can grow to infinity, we can partition the set W into sets of size $\epsilon \Delta 7^l$ and add cliques on them. This will ensure that majority is satisfied for all vertices in W. The claims and the proofs of this theorem are practically identical to those in section 3.2. We omit them for want of space.

5 Regular Graphs

Proposition 1. *If G is a regular graph then any monopoly solution in G has size at least $\frac{1}{2}|V(G)|$.*

Proof. Let G be D-regular, and let $X \subset V(G)$ be a monopoly on G. Then, each vertex of G outside X has at least $(D+1)/2$ neighbours in X and each vertex in X has at least $(D+1)/2 - 1$ neighbours in X. Thus,

$$D \cdot |X| \;\geq\; \left(\frac{D+1}{2}\right) \cdot |V(G)| - |X|.$$

Proposition 2. *If G is D-regular that any partial monopoly solution in G has size at least $\left(\frac{D+1}{3D+1}\right) \cdot |V(G)|$.*

Proof. Let $X \subseteq V(G)$ be a partial monopoly in G. Then, each vertex of G outside X has at least $(D+1)/2$ neighbours in X. Thus,

$$D \cdot |X| \;\geq\; \left(\frac{D+1}{2}\right) \cdot (|V(G)| - |X|).$$

These propositions show that there are trivial constant factor approximation algorithms for the monopoly and partial monopoly problems if the graph is regular. In particular, for 3 regular graphs, we have a factor 2.5 algorithm for the partial monopoly problem and a factor 2 algorithm for the monopoly problem. Below, we improve on these trivial bounds. On the other hand, we show that the problem of approximating the minimum monopoly and the minimum partial monopoly for 3-regular case is APX-complete; so unless $\mathsf{P} = \mathsf{NP}$ there cannot be a PTAS for these problems.

Proof of Theorem 3(a)

Proof. Given a 3-regular graph $G = (V, E)$, we will produce two monopolies in it and take the one of smaller size as our solution.

1. Let M be a matching in G. Delete all edges in M. Let the resulting graph be G'. Note that each vertex has at least two neighbours in G'. For each vertex v, pick one pair of neighbours p_v. Let H be the graph obtained from G by adding all pairs p_v to the edge set of G'. That is,
$$E(H) = (E(G) - M) \cup \{p_v : v \in V\}.$$
Let I be an independent set in H. Since, $p_v \cup \{v\}$ induces a triangle in H, I can include at most one vertex from this set. Thus, $V - I$ is a monopoly in G.
To get a monopoly of small size, we first pick a matching of size as large as possible. Let the matching have αn edges, where $|V(G)| = n$. Note, that the number of edges in H is at most $\frac{3}{2}n - \alpha n + n$. In H, find a large independent set using the greedy algorithm [6,7]. Since the average degree H is at most $5 - 2\alpha$, the greedy algorithm return an independent set of size at least $\frac{n}{6-2\alpha}$ [6]. Thus, the monopoly $V - I$ has size at most $n(1 - \frac{1}{6-2\alpha})$.
2. Let M be a maximal matching. Then, the set of vertices matched by M is a monopoly in G. If M, has αn edges, then this monopoly has $2\alpha n$ vertices.

It can be verified that $\min\{n(1 - \frac{1}{6-2\alpha}), 2\alpha n\}$ is maximum when $\alpha = \frac{7-\sqrt{29}}{4}$. In this case, the size of our solution is at most $2\alpha n$. Since the optimum solution has size at least $\frac{n}{2}$ (by Proposition 1), this is bigger than the optimum by a factor of at most $7 - \sqrt{29} \leq 1.6154$. □

Theorem 7. *For 3-regular graphs, monopoly problem is APX-complete.*

Proof. Since, for 3-regular graphs, monopoly problem is in the class APX and vertex cover problem is APX-complete [1], it is enough to establish a L-reduction [13] from vertex cover problem to monopoly problem.

Given a 3-regular graph $G = (V, E)$, an instance of vertex cover problem, construct another 3-regular graph $G' = (V', E')$, an instance of monopoly problem, as follows:

1. Make two copies $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ of $G = (V, E)$, ie. $V_i = \{v_i | v \in V\}$ and $E_i = \{(u_i, v_i) | (u, v) \in E\}$, for $i = 1, 2$.
2. For each edge $(u, v) \in E$, attach the graph $H(u, v)$ to the graphs G_1 and G_2 as shown in the Figure 1. $H(u, v)$ is the graph within the dotted line as shown in Figure 1.

Clearly, G' is 3-regular and can be constructed from G in polynomial time. First, we state some claims which are easy to prove.

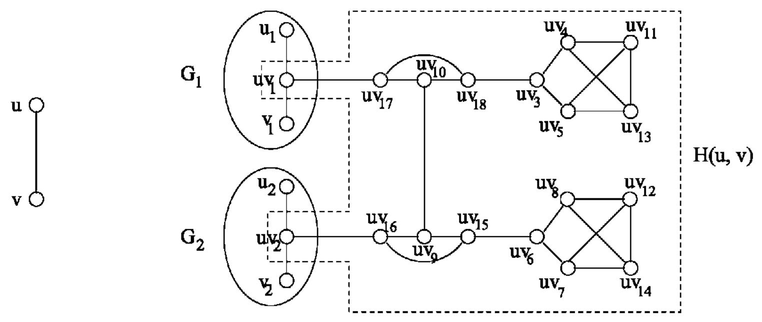


Fig. 1. An edge (u, v) in G and its counter part in G'

Claim. Consider the subgraph of $H(u, v)$ induced by the vertex set $\{uv_3, uv_4, uv_5, uv_{11}, uv_{13}\}$. At least 3 vertices are required to satisfy the majority requirement at all these vertices; and without loss of generality, we assume that a minimal monopoly set X of G' contains the vertices uv_3, uv_4, uv_5 out of these 5 vertices.

Claim. In addition to the vertex set $\{uv_3, uv_4, uv_5, uv_6, uv_7, uv_8\}$, at least 4 more vertices are required to satisfy the majority requirement at all the vertices of $H(u, v)$; and we assume that these 4 vertices are uv_1, uv_2, uv_9 and uv_{10}.

Let $T = \cup_{(u,v)\in E}\{uv_1, uv_2, \ldots, uv_{10}\}$. On the basis of both the above claims, without loss of generality, we assume that a minimal monopoly set X of G' contains the set T as a proper subset. It is easy to see that $|T| = 15n$.

For a given minimal vertex cover S of G, the set $X = S_1 \cup S_2 \cup T$ is a minimal monopoly set of G', where $S_i = \{v_i | v \in S\}$, for $i = 1, 2$. It is easy to see that $|N_{G'}[p] \cap X| \geq 2$ at each vertex $p \in V' - \{uv_1, uv_2 | (u, v) \in E\}$. Also, $|N_{G'}[p] \cap X| \geq 2$, for $p \in \{uv_1, uv_2 | (u, v) \in E\}$, as $N_{G'}[uv_i] \cap X$ contains the vertex uv_i and at least one of u_i and v_i (because S_i is a vertex cover of G_i), for $i = 1, 2$. Moreover, X is a minimal monopoly of G'. This is because, no vertex from T can be dropped (by the two claims). This is because no vertex can be dropped from $X - T$ as S is a minimal vertex cover of G. Note that $|X| = 15n + 2|S|$.

To a minimal monopoly X of G' we associate a set $S = \{v \in V | v_1 \in V_1 \cap X\}$. S is a vertex cover of G as, for each vertex $uv_1 \in X \cap T$, $N_{G'}[uv_1]$ contains either u_1 or v_1 and is true for each edge $(u, v) \in E$. Also $|X| = 15n + 2|S|$.

From these two observations, it is now easy to see that S_o is a minimum vertex cover of G if and only if X_o is a minimum monopoly of G'. Also $|X_o| = 15n + 2|S_o| \leq 2|S_o| + 60|S_o| = 62|S_o|$ (as $n \leq 4|S_o|$). For any minimal monopoly X of G', we have $|X_o| - |X| = 2|S_o| - 2|S|$. This establish a L-reduction with $\alpha = 62$ and $\beta = \frac{1}{2}$. □

Proof of Theorem 3(b)

Proof. For any partial monopoly set X of a 3-regular graph G, a connected component in the induced subgraph $G[V - X]$ is either an edge or an isolated vertex. This is because, in $G[V - X]$ no vertex can have degree more than 1.

Hence, we will construct a maximal vertex set T such that each connected component in $G[T]$ is an edge and return $X = V - T$ as a partial monopoly of G. We construct such a set T, starting from an empty set T, step by step by adding a pair of vertices at a time and deleting at most 6 vertices from G till G has no edges. In each step, we choose an edge (u, v) in G, update T by $T \cup \{u, v\}$ and then remove the vertex set $N[u] \cup N[v]$ from G along with all the edges incident on them.

Here, at each step we add 2 vertices and remove at most 6 vertices from G. Hence, $|T| \geq \frac{2}{6}n$. Thus, $|X| \leq \frac{4}{6}n$. But by Proposition 2, we know $|X_o| \geq \frac{2}{5}n$ where X_o is a minimum partial monopoly set of G. Hence, $\frac{|X|}{|X_o|} \leq \frac{5}{3}$. □

Theorem 8. *For 3-regular graphs, partial monopoly problem is APX-complete.*

Proof. Since, for 3-regular graphs, partial monopoly problem is in the class APX, it is enough to establish a L-reduction from vertex cover problem to partial monopoly problem.

Given a 3-regular graph $G = (V, E)$, an instance of vertex cover problem, construct a 3-regular graph $G' = (V', E')$, an instance of partial monopoly problem, as follows:

1. Make two copies $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ of $G = (V, E)$, ie. $V_i = \{v_i | v \in V\}$ and $E_i = \{(u_i, v_i) | (u, v) \in E\}$, for $i = 1, 2$.
2. For each edge $(u, v) \in E$, attach the graph $H(u, v)$ to the graphs G_1 and G_2 as shown in the Figure 2. $H(u, v)$ is the graph within the dotted line as shown in Figure 2.

It is easy to see that G' is 3-regular and can be constructed from G in polynomial time.

Claim. Consider the subgraph of $H(u, v)$ induced by the vertex set $\{uv_1, uv_2, uv_3, uv_4, uv_{11}, uv_{12}\}$. At least 4 vertices are required to satisfy the partial monopoly requirement at these vertices and without loss of generality we assume that a minimal partial monopoly set X of G' contains the four vertices uv_1, uv_2, uv_3, uv_4, for each $(u, v) \in E$.

Claim. Let S be a minimal vertex cover of G. Then the set X is a partial monopoly set of G', where

$$X = S_1 \cup S_2 \cup \left[\bigcup_{(u,v) \in E} \{uv_1, uv_2, uv_3, uv_4\} \right] \cup \left[\bigcup_{(u,v) \in E; u \in S; v \notin S} \{uv_5, uv_6\} \right]$$
$$\cup \left[\bigcup_{(u,v) \in E; u \notin S; v \in S} \{uv_7, uv_8\} \right] \cup \left[\bigcup_{(u,v) \in E; u \text{ and } v \in S} \{uv_7, uv_8\} \right],$$

and $S_i = \{v_i | v \in S\}$, for $i = 1, 2$. Also $|X| = 2|S| + 9n$.

Proof of claim: In order to prove this claim we need to show that majority requirement is satisfied only at vertices in $V' - X$. We omit this check.

As X contains exactly 6 vertices from $H(u, v)$, for each $(u, v) \in E$, we have $|X| = 2|S| + 6\frac{3}{2}n = 2|S| + 9n$. This completes the proof of *Claim*.

For the rest of the proof of this theorem, we assume that a minimal partial monopoly set X of G' does not contain the vertices of the form uv_9 and uv_{10}, for all $(u, v) \in E$. Otherwise we can construct another minimal partial monopoly set X' not containing uv_9 and uv_{10} with $|X'| \leq |X|$. To see this, let X be a minimal partial monopoly of G' containing one such vertex, say the vertex uv_9, for some edge $(u, v) \in E$. Clearly, the set $X \cup \{uv_7\} - \{uv_9\}$ is a partial monopoly set of G' and let X' be a minimal partial monopoly set contained in this new set. Similar arguments apply for the vertex uv_{10}.

Throughout this proof, we assume that a minimal partial monopoly set X of G' does not contain both the vertices uv_6 and uv_7, for any $(u, v) \in E$. This is because the set $X \cup \{u_1\} - \{uv_7\}$ is a partial monopoly set of G' and the minimal partial monopoly set of G' contained in this set is of size at most $|X|$. Similarly, we assume that any minimal partial monopoly set of G' does not contain both the vertices uv_5 and uv_8, for any $(u, v) \in E$.

Hence, we will consider those minimal partial monopolies which are having exactly 6 vertices from $H(u, v)$, for each $(u, v) \in E$.

Claim. Let X be a minimal partial monopoly of G' which contains exactly 6 vertices from $H(u, v)$, for each $(u, v) \in E$. Then $S = \{v | v_1 \in X \cap V_1\}$ is a vertex cover of G and $|X| = 2|S| + 9n$.

Proof of claim: Since $|H(u, v) \cap X| = 6$, X contains either uv_6 or uv_7, for each $(u, v) \in E$. If $uv_6 \in X$ then $u_1 \in S$ (because $N_{G'}[uv_7] \cap X$ contains at least two vertices and both uv_7 and uv_9 are not in X). Similarly, if $uv_7 \in X$ then $v_1 \in X$.

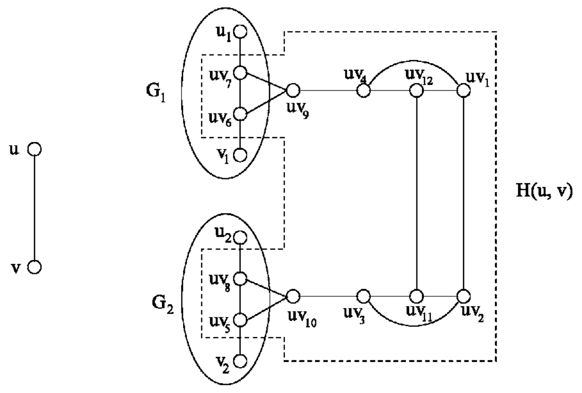


Fig. 2. An edge (u, v) in G and its counter part in G'

This is true for each edge $(u, v) \in E$. Hence, S as defined is a vertex cover of G. The proof of the remaining part of this claim is easy.

From the previous two claims, it is easy to show that S_o is a minimum vertex cover of G if and only if X_o (as defined in the earlier claim) is a minimum partial monopoly of G'. Thus $|X_o| = 2|S_o| + 9n \leq 2|S_o| + 36|S_o| = 38|S_o|$. Also, for each minimal pmonopoly X of G', we have $|X| - |X_o| = 2\,(|S| - |S_o|)$. This shows that this is an L-reduction with $\alpha = 38$ and $\beta = \frac{1}{2}$. □

References

1. P. Alimonti AND V. Kann, Hardness of approximating problems on cubic graphs, in *Proc. 3rd Italian Conf. on Algorithms and Complexity*, LNCS-1203, Springer-Verlag, 1997, 288-298.
2. G Bracha, An $o(\log n)$ expected rounds randomised Byzantine generals algorithm. *Journal of the ACM*, 1987, 910-920.
3. U. Feige, A threshold of $\ln n$ for approximating set cover, In *Proc. ACM Symp. on the Theory of Computing*, 1996.
4. P. Flocchini, F. Geurts AND N. Santoro, Dynamic majority in general graphs and chordal rings (Draft) 1997.
5. R. Greenlaw AND R. Petreschi, Cubic graphs, *ACM Computing Surveys*, vol. 27, 1995, 471-495.
6. D. S. Hochbaum, Efficient bounds for the stable set, vertex cover and set packing problems, *Disc. Appl. Math.*, vol. 6, 1982, 243-254.
7. M. Halldórsson AND J. Radhakrishnan, Greed is Good: Approximating Independent Sets in Sparse and Bounded-Degree Graphs, *Algorithmica*, vol. 18, 1997, 145-163.
8. S. Kutten AND D. Peleg, Fault-local distributed mending. In *Proc. 36th IEEE Symp. on Foundations of Computer Science* 1995.
9. L. Lamport, R. Shostak AND M. Pease, The Byzantine generals problem. *ACM Transactions on Programming Languages and systems*, vol. 4, 1982, 382-401.
10. C. Lund AND M. Yannakakis, On the hardness of Approximating minimisation problems. *Journal of ACM*, vol. 41, 1994, 960-981.
11. D. Peleg, Local majority voting, small coalition and controlling monopolies in graphs: A review. In *Proc. of 3rd Colloquium on Structural Information and Communication Complexity*, 152-169, 1996.
12. Luca Trevisan, Non-approximability Results for Optimization Problems on Bounded Degree Instances. *In Proc. of the 33rd ACM STOC*, 2001.
13. C. H. Papadimitriou AND M. Yannakakis, Optimization, Approximation, and Complexity Classes. *J. Comput. System Sci.*, vol. 43, 1991, 425-440.
14. S. Ueno, Y. Kajtani AND S. Gotoh, On the non-separating independent set problem and feedback set problem for graphs with no vertex exceeding three, *Disc. Math.*, vol. 72, 1988, 355-360.
15. V. Vazirani, *Approximation Algorithms*, Springer, 2001.

Hereditary History Preserving Bisimulation Is Decidable for Trace-Labelled Systems[⋆]

Madhavan Mukund

Chennai Mathematical Institute, 92 GN Chetty Road, Chennai 600017, India
madhavan@cmi.ac.in

Abstract. Hereditary history preserving bisimulation is a natural extension of bisimulation to the setting of so-called "true" concurrency. Somewhat surprisingly, this extension turns out to be undecidable, in general, for finite-state concurrent systems. In this paper, we show that for a substantial and useful class of finite-state concurrent systems – those whose semantics can be described in terms of Mazurkiewicz traces – hereditary history preserving is decidable.

1 Introduction

While branching time semantics is relatively well understood in the interleaved approach to the semantics of concurrent systems, the situation is not so clear when labelled partial orders are used to record the behaviour of such systems. If no restriction is placed on the structure of a concurrent system, the interplay between nondeterminism and concurrency results in many seemingly simple problems becoming computationally intractable.

An example of this is the problem of checking whether two finite-state systems are equivalent with respect to the concurrency-preserving branching-time behavioural equivalence known as hereditary history preserving bisimulation. Hereditary history preserving bisimulation is a natural extension of bisimulation, as defined by Park and Milner [11,13], from a setting where concurrency is equated with nondeterministic interleaving to a richer setting where nondeterminism and concurrency are represented explicitly and independently. Hereditary history preserving bisimulation was first defined by Bednarczyk [1], but became more well known when it reappeared as a natural construction in a general, categorical approach to nondeterminism and concurrency arising out of the work of Winskel and Nielsen[7,17].

Hereditary history preserving bisimulation requires two concurrent systems to retain the same nondeterministic choices as they evolve, as in conventional bisimulation. In addition, at every state, each of the systems also has to faithfully simulate all steps – sets of pairwise independent actions – performed by other system. Although this seems to be a fairly innocuous extension, the repercussions are quite severe. It turns out that history preserving bisimulation is, in general, undecidable for finite-state concurrent systems [8].

[⋆] Partially supported by IFCPAR Project 2102-1 (ACSMV).

A few positive results have been obtained regarding hereditary history preserving bisimulation. In [12], a game-theoretic formulation is presented, along with a characterization in terms of a Hennessy-Milner style modal logic with past modalities. The decidability question was investigated in [5] where some very restricted positive results were obtained.

Earlier, a weaker notion of history preserving bisimulation had been proposed in [3,15,16]. Here, too, the two systems have to progressively simulate each others' concurrent steps. However, the bisimulation is built up one action at a time, so each interleaving of a concurrent step in one system may be simulated by a different, incompatible, step in the other system. Thus, from the standpoint of faithfully preserving concurrency and nondeterminism, this notion is slightly unsatisfactory. The decidability of this variety of bisimulation was established in [6].

In this paper, we examine the decidability question afresh for a restricted class of concurrent systems – those whose behaviours can be described by Mazurkiewicz traces. Our main result is that hereditary history preserving bisimulation is decidable for this class of systems.

Mazurkiewicz traces [10] are labelled partial orders generated by independence alphabets of the form (Σ, I), where I is a static *independence* relation over Σ. If $(a, b) \in I$, a and b are deemed to be independent actions that may occur concurrently in any context where they are jointly enabled. Traces are a natural formalism for describing the behaviour of various static networks of communicating finite-state agents as modelled by Petri nets [14] or communicating finite-state automata [18]. Hence, our positive result is applicable to a substantial and useful subclass of finite-state concurrent systems.

The paper is organized as follows. We begin with some basic definitions about labelled Petri nets, the system model that we work with in this paper. In Section 3 we define hereditary history preserving bisimulation. In the next section, we identify the subclass of systems that we focus on – those whose semantics can be defined using Mazurkiewicz traces. In Section 5, we show that hereditary history preserving bisimulation is equivalent to a notion of step bisimulation on step transition systems. This characterization is used to derive the main decidability result in Section 6. We conclude with a brief discussion.

2 Preliminaries

We use labelled 1-safe Petri nets as our basic model of finite-state concurrent systems. This choice of model is not important: we could have, instead, worked with any other model that has an explicit notion of independence or concurrency built in, such as labelled asynchronous transition systems [2], asynchronous automata [18] or transition systems with independence [17].

Nets. A *net* is a quadruple (S, T, F, M_{in}) where:

- S is a finite, non-empty set of *places.*
- T is a finite, non-empty set of *transitions.*

– $F \subseteq (S \times T) \cup (T \times S)$ is the *flow relation.*
– $M_{in} : S \to \mathbb{N}_0$ is the *initial marking*, where $\mathbb{N}_0 = \{0, 1, 2, \ldots\}$.

As usual, for x in $S \cup T$, we use ${}^\bullet x$ to denote the set $\{y \in S \cup T \mid (y, x) \in F\}$ and $x^\bullet$ to denote the set $\{y \in S \cup T \mid (x, y) \in F\}$.

Reachable markings. A *marking* of (S, T, F, M_{in}) is a function $M : S \to \mathbb{N}_0$ and corresponds to a global state of the system. A transition t is enabled at marking M, denoted $M \xrightarrow{t}$, if $M(s) > 0$ for all $s \in {}^\bullet t$. When t occurs, M evolves to a new marking M', written $M \xrightarrow{t} M'$, where

$$\forall s \in S.\ M'(s) = \begin{cases} M(s) - 1 \text{ if } s \in ({}^\bullet t - t^\bullet), \\ M(s) + 1 \text{ if } s \in (t^\bullet - {}^\bullet t), \\ M(s) \qquad \text{otherwise.} \end{cases}$$

Notice that M' is uniquely fixed by M and t. Let $w = t_1 t_2 \ldots t_k$. We write $M \xrightarrow{w} M'$ to indicate that $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \cdots M_{k-1} \xrightarrow{t_k} M'$. Similarly, $M \xrightarrow{w}$ denotes that there exists M' such that $M \xrightarrow{w} M'$.

Let M_0 be a marking of (S, T, F, M_{in}). The set of markings reachable from M_0, denoted $\mathcal{R}(M_0)$ is defined inductively as follows:

– $M_0 \in \mathcal{R}(M_0)$.
– If $M \in \mathcal{R}(M_0)$ and $M \xrightarrow{t} M'$, then $M' \in \mathcal{R}(M_0)$.

1-safe nets. The net (S, T, F, M_{in}) is said to be *1-safe* if for all $M \in \mathcal{R}(M_{in})$, for all $s \in S$, $M(s) \leq 1$. Clearly, if N is 1-safe, then its global state space is finite since the number of distinct markings in $\mathcal{R}(M_{in})$ is bounded by $2^{|T|}$. In this paper, we assume that every net we consider is 1-safe.

Labelled nets. Let Σ be a set of actions. A Σ-labelled net is a structure $N = (S, T, F, M_{in}, \lambda)$ where (S, T, F, M_{in}) is a (1-safe) net and $\lambda : T \to \Sigma$ is a labelling function.

Independence of transitions. We say that transitions t_1 and t_2 are *independent* if they have disjoint neighbourhoods – that is, $({}^\bullet t_1 \cup t_1{}^\bullet) \cap ({}^\bullet t_2 \cup t_2{}^\bullet) = \emptyset$. We write $t_i I_N t_j$ to denote that t_i and t_j are independent in N. If $\neg(t_i I_N t_j)$ we write $t_i D_N t_j$, denoting that t_i and t_j are *dependent*. Independent transitions satisfy forward and sideways diamond properties. Let t_1 and t_2 be independent. If $M \xrightarrow{t_1} M_1$ and $M \xrightarrow{t_2} M_2$ then there exists M' such that $M_1 \xrightarrow{t_2} M'$ and $M_2 \xrightarrow{t_1} M'$. Further, if $M \xrightarrow{t_1} M_1 \xrightarrow{t_2} M'$ then there exists M_2 such that $M \xrightarrow{t_2} M_2 \xrightarrow{t_1} M'$.

Runs. Let $N = (S, T, F, M_{in}, \lambda)$ be a labelled net. Each transition sequence $w = t_1 t_2 \ldots t_k$ such that $M_0 \xrightarrow{w} M_w$ gives rise to a labelled partial order that we call a *run.* The run $po(w)$ associated with w is a triple $(E, \leq, \tau)$ where E is a set of events partially ordered by $\leq$ and $\tau : E \to T$ is a labelling function. Each event in E corresponds to a transition from w and is labelled by the underlying transition. The partial order $\leq$ is the reflexive, transitive closure of the relation $e_i \prec e_j \Leftrightarrow i < j \wedge t_i D_N t_j$, where $\tau(e_i) = t_i$ and $\tau(e_j) = t_j$.

More formally, let $w = t_1 t_2 \ldots t_k$. Then, $po(w) = (E, \leq, \tau)$ such that:

(i) $E = \{1, 2, \ldots, k\}$.
(ii) For each $i \in E$, $\tau(i) = t_i$.
(iii) For $i, j \in E$, if i and j are unordered in $po(w)$ then $t_i I_N t_j$.
(iv) For $i, j \in E$, if $i \lessdot j$ then $t_i D_N t_j$, where $\lessdot \; = \; < \setminus <^2$ is the immediate successor relation in $po(w)$.

Let $\mathit{Runs}(N) = \{po(w) \mid w \in T^*, M_{in} \xrightarrow{w}\}$ denote the set of runs of N. With each run $r \in \mathit{Runs}(N)$, we can associate a unique marking M_r, the global state of N after the computation r.

If $r = po(w)$ and $r' = po(wt)$ – that is, r' extends r by adding an event corresponding to the transition t – we denote r' as $r + t$. If $u = \{t_1, t_2, \ldots, t_k\}$ is a set of pairwise independent transitions enabled at M_r, then we denote the run $r + t_1 + t_2 + \cdots + t_k$ by $r + u$.

For a run $r = (E, \leq, \tau)$, $\max(r)$ denotes the set of maximal events in E. Let $r = po(w)$, where $w = t_1 t_2 \ldots t_k$, and let $j \in \max(r)$. Then $r - j$ denotes the run obtained by deleting the event j from r – that is, $r - j = po(w')$, where $w' = t_1 t_2 \ldots t_{j-1} t_{j+1} \ldots t_k$.

Two runs are said to be isomorphic if they are isomorphic as labelled partial orders. We write $r \simeq r'$ to indicate that r and r' are isomorphic runs.

For each net, the empty sequence of transitions ε gives rise to the empty run $(\emptyset, \emptyset, \emptyset)$. We use $r_\emptyset$ uniformly to denote the empty run for all nets.

3 History Preserving Bisimulations

For the rest of this section, fix a pair of labelled nets $N_i = (S_i, T_i, F_i, M^i_{in}, \lambda_i)$, $i \in \{1, 2\}$, whose transitions are labelled by a common set of actions Σ. A history preserving bisimulation is a relation between the runs of N_1 and the runs of N_2 which asserts that the two systems have equivalent observable capabilities, even when we take concurrency into account.

We define history preserving bisimulations in terms of matched runs.

Matched runs. Let $\rho = (E, \leq, \tau_1, \tau_2)$ be a partial order equipped with two labelling functions such that $\tau_i : E \to T_i$, $i \in \{1, 2\}$, labels each event in E with a transition from T_i. The structure $(E, \leq, \tau_1, \tau_2)$ is a *matched run* of N_1 and N_2 if it satisfies the following conditions:

- $\rho_1 = (E, \leq, \tau_1)$ is a run of N_1 and $\rho_2 = (E, \leq, \tau_2)$ is a run of N_2.
- For all $e \in E$, $\lambda_1(\tau_1(e)) = \lambda_2(\tau_2(e))$.

Let ρ and ρ' be matched runs such that ρ' extends ρ by k events with $\rho'_1 = \rho + u_1$ and $\rho'_2 = \rho_2 + u_2$, where u_1 and u_2 are pairwise disjoint subsets of size k of T_1 and T_2, respectively. We denote ρ' by $\rho + \langle u_1, u_2 \rangle$. If $u_1 = \{t_1\}$ and $u_2 = \{t_2\}$ are singletons, we write $\rho + \langle t_1, t_2 \rangle$ rather than $\rho + \langle \{t_1\}, \{t_2\} \rangle$.

Let e be a maximal event in the matched run ρ. Then $\rho' = \rho - e$ is the matched run obtained by deleting e from ρ. If $u = \{e_1, e_2, \ldots, e_k\}$ is a subset of maximal events in ρ, we write $\rho - u$ to denote the run $(((\rho - e_1) - e_2) - \cdots - e_{k-1}) - e_k$

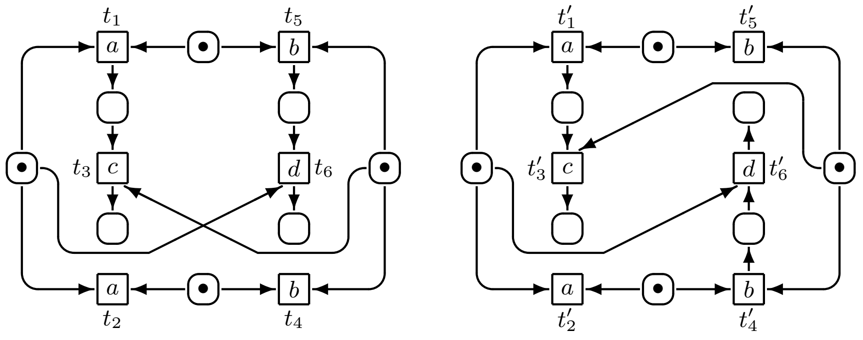


Fig. 1.

obtained by deleting all the events from u (clearly, the events in u may be removed in any order).

Hereditary history preserving bisimulation. A *hereditary history preserving bisimulation* between $N_1 = (S_1, T_1, F_1, M^1_{in}, \lambda_1)$ and $N_2 = (S_2, T_2, F_2, M^2_{in}, \lambda_2)$ is a set H of matched runs of the two nets such that:

(i) The empty matched run $(\emptyset, \emptyset, \emptyset, \emptyset)$ belongs to H.
(ii) Let $\rho = (E, \leq, \tau_1, \tau_2) \in H$. For each transition $t_1 \in T_1$ enabled at M_{ρ_1}, there is a transition t_2 in T_2 such that $\rho + \langle t_1, t_2 \rangle \in H$.
(iii) Let $\rho = (E, \leq, \tau_1, \tau_2) \in H$. For each transition $t_2 \in T_2$ enabled at M_{ρ_2}, there is a transition t_1 in T_1 such that $\rho + \langle t_1, t_2 \rangle \in H$.
(iv) For each maximal event e in ρ, the matched run $\rho - \{e\}$ is in H.

The first three clauses correspond to the weaker definition of history preserving bisimulations [3,15,16]. The last clause strengthens the definition by ensuring that the bisimulation extends concurrent steps in a uniform way, regardless of the order in which the concurrent step is executed. The weaker definition permits different interleavings of the same concurrent step to be simulated in different ways. (The original definition of hereditary history preserving bisimulation required all prefixes of a matched run to belong to the relation H. However, it was shown in [12] that is suffices to check the condition for the substructures obtained by deleting each of the maximal events in the given matched run.) Figure 1, taken from [5], illustrates the difference between the two definitions.

In this example, consider the step $\{t_1, t_4\}$ in the net on the left. If we execute t_1 before t_4, we can simulate this by t'_1 followed by t'_4, preserving forward choices at each point. Similarly, if we execute t_4 before t_1, we can simulate it by t'_5 followed by t'_2. However, neither simulation is faithful to the original step and this is caught by the last clause in the definition of hereditary history preserving bisimulations – for instance, if we simulate $t_1 t_4$ by $t'_1 t'_4$, we can "undo" the maximal events t_1 and t'_1 to reach markings in which a d-labelled transition is enabled on the right but not on the left.

The difficulty with establishing whether two labelled nets are hereditary history-preserving bisimilar is that the bisimulation is defined at the level of matched runs, which correspond to the infinite "unfolded" behaviours of the

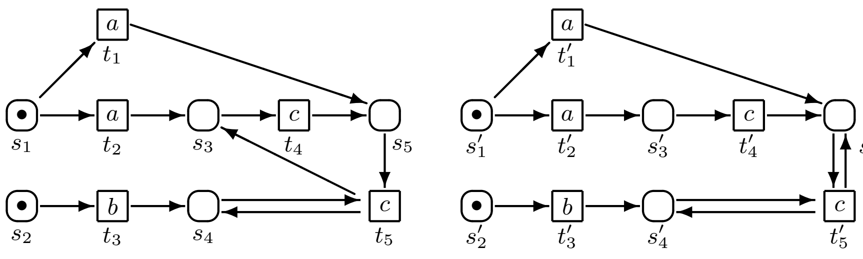


Fig. 2.

two nets. For normal sequential bisimulation, given a pair of finite transition systems, a bisimulation relation exists between their unfoldings if and only if a (finite) bisimulation relation exists between the states of the original system.

Every Petri net can be regarded as a labelled transition system whose states correspond to the (reachable) markings of the net. In general, a hereditary history preserving bisimulation at the level of matched runs cannot be "folded" down to a relation the level of markings, as demonstrated in Figure 2. These two nets are hereditary history preserving bisimilar. The markings $\{s_3, s_4\}$ and $\{s'_4, s'_5\}$ are not equivalent after executing $\{t_2, t_3\}$ and $\{t'_1, t'_3\}$. However, if we reach the same markings after executing $\{t_2, t_3, t_4, t_5\}$ and $\{t'_2, t'_3, t'_4, t'_5\}$, the markings do become equivalent.

4 Trace-Labelled Systems

Trace alphabets. A *trace alphabet* is a pair (Σ, I), where $I \subseteq \Sigma \times \Sigma$ is an irreflexive and symmetric *independence relation*. The complement of I is denoted D and called the *dependence relation*.

The independence relation I induces a natural equivalence relation $\sim$ on words over Σ. Intuitively, two words are related by $\sim$ if we can go from one to another by repeatedly swapping adjacent independent letters. More formally, let $\sim_0$ be the relation $\{(wabw', wbaw') \mid w, w' \in \Sigma^*, aIb\}$. Then $\sim$ is the reflexive, transitive closure of $\sim_0$. The equivalence classes generated by $\sim$ are called *(Mazurkiewicz) traces*. We denote the trace containing the word w by $[w]$.

Trace-labelled nets. Let (Σ, I) be a trace alphabet. A *labelled net* $N = (S, T, F, M_{in}, \lambda)$ with $\lambda : T \to \Sigma$ is said to be *trace-labelled* if $t_i I_N t_j \Leftrightarrow \lambda(t_i) I \lambda(t_j)$ for all pairs of transitions $t_i, t_j \in T$.

For trace-labelled nets, the partial order structure of a run is determined solely by the labelling on the underlying transition sequence. Formally, we have the following result.

Proposition 1. *Let $N = (S, T, F, M_{in}, \lambda)$ be a trace-labelled net and let $\rho(w)$ and $\rho(w')$ be two runs of N such that $w = t_1 t_2 \ldots t_k$, $w' = t'_1 t'_2 \ldots t'_k$ and $\lambda(t_i) = \lambda(t'_i)$ for each $i \in \{1, 2, \ldots, k\}$. Then, $\rho(w) \simeq \rho(w')$.*

We omit the proof this result, which is a reformulation of a standard result of trace theory that any linearization of a trace fixes its underlying partial order representation [4].

$$\begin{tikzcd}
 & & M_5 & & M_9 \\
 & M_1 \arrow[ur, "a"] \arrow[dl, "a"'] & & M_2 \arrow[ul, "b"'] \arrow[ur, "d"] \arrow[dr, "b"] & \\
M_6 \arrow[ur, bend left, "e"] & & M_{in} \arrow[ul, "b"] \arrow[ur, "a"] \arrow[dl, "a"] \arrow[dr, "b"] & & M_7 \arrow[ulll, bend right, "e"'] \\
 & M_3 \arrow[ul, "b"] \arrow[dr, "b"] & & M_4 \arrow[ur, "a"] \arrow[dl, "a"] & \\
 & & M_8 & &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & & M'_5 & & \\
 & M'_1 \arrow[ur, "a"] \arrow[dl, "a"'] & & M'_2 \arrow[ul, "b"'] \arrow[dr, "b"] & \\
M'_6 \arrow[ur, bend left, "e"] & & M'_{in} \arrow[ul, "b"] \arrow[ur, "a"] \arrow[dl, "a"] \arrow[dr, "b"] & & M'_7 \arrow[ulll, bend right, "e"'] \\
 & M'_3 \arrow[ul, "b"] \arrow[dl, "d"] \arrow[dr, "b"] & & M'_4 \arrow[ur, "a"] \arrow[dl, "a"] & \\
M'_9 & & M'_8 & &
\end{tikzcd}$$

Fig. 3.

Observe that the nets in Figure 2 are not trace-labelled. For instance, in the net on the left, the transition sequences $t_1t_3t_5$ and $t_2t_3t_4$ both generate the same labelled sequence, *abc*, but give rise to non-isomorphic runs.

In Figure 2, one reason that we could not fold down the hereditary history preserving bisimulation relation to a relation on markings was because the independence between actions b and c is context-dependent. This problem is eliminated if we work with trace-labelled nets. Nevertheless, even for trace-labelled nets, we cannot always translate a hereditary history preserving bisimulation relation into a relation on markings. Figure 3 shows the reachable markings of a pair of trace-labelled nets that are hereditary history preserving bisimilar, where aIb. Here, if we initially execute $M_{in} \xrightarrow{b} M_1$ and $M'_{in} \xrightarrow{b} M'_1$, the markings are not equivalent, but after the sequences $M_{in} \xrightarrow{a} M_2 \xrightarrow{b} M_7 \xrightarrow{e} M_1$ and $M'_{in} \xrightarrow{a} M'_3 \xrightarrow{b} M'_6 \xrightarrow{e} M'_1$, the markings are in fact equivalent.

5 Synchronized Step Transition Systems

Step transition systems. Let (A, I) be a trace alphabet. A *step transition system* over (A, I) is a structure $TS = (Q, \delta, q_{in})$ where Q is a set of states with an initial state $q_{in} \in Q$ and $\delta : Q \times 2^A \to Q$ is a step transition function satisfying the following conditions:

(i) If $\delta(q, u) = q'$, then for each distinct pair $a_i, a_j \in u$, a_iIa_j.
(ii) If $\delta(q, u) = q'$, then for all $v \subseteq u$, there exists q_v such that $\delta(q, v) = q_v$ and $\delta(q_v, u \setminus v) = q'$.

Thus, transitions in a step transition system are labelled by sets of pairwise independent actions in such a way that every step can be broken up into all possible substeps. This definition of step transition systems is equivalent to *deterministic* distributed transition systems, as defined in [9].

Notice that for independent actions a and b, it is possible to have transitions $\delta(q,a) = q_a$, $\delta(q,b) = q_b$ and $\delta(q_a,b) = \delta(q_b,a) = q'$, but *not* have the step transition $\delta(q,\{a,b\}) = q'$. A step transition system that does not exhibit such anomalies is said to be coherent.

Coherent step transition systems. A step transition system $TS = (Q, \delta, q_{in})$ over (A, I) is said to be coherent if the following holds: whenever $\delta(q,a) = q_a$ and $\delta(q_a, b) = q'$ for aIb, it is the case that $\delta(q,\{a,b\}) = q'$.

Proposition 2. *Let $TS = (Q, \delta, q_{in})$ be a coherent step transition system over (A, I) and let $q \in Q$ be a state that is reachable from the initial state q_{in}. Let $w \in A^*$ be a word such that $\delta(q_{in}, w) = q$. For every $w' \sim w$, $\delta(q_{in}, w') = q$ (recall that $\sim$ is the trace equivalence relation induced by I).*

The proof, which we omit, is by induction on the number of times we have to apply the immediate trace equivalence $\sim_0$ to go from w to w'.

Synchronous step transition systems. Let $N_i = (S_i, T_i, F_i, M^i_{in}, \lambda_i)$, $i \in \{1,2\}$ be a pair of labelled Petri nets. We can extend the independence relations I_{N_1} on T_1 and I_{N_2} on T_2 to an independence relation $I_{N_{12}}$ on $T_1 \times T_2$ as follows: $(t_1, t_2) I_{N_{12}} (t'_1, t'_2)$ if and only if $t_1 I_{N_1} t'_1$ and $t_2 I_{N_2} t'_2$.

A *joint step transition system* for N_1 and N_2 is a step transition system $TS = (Q, \delta, q_{in})$ over $(T_1 \times T_2, I_{N_{12}})$ where:

- $Q = \{(r_1, r_2) \mid r_1 \in \mathit{Runs}(N_1), r_2 \in \mathit{Runs}(N_2), r_1 \simeq r_2\}$.
- $q_{in} = (r_\emptyset, r_\emptyset)$ (recall that for any net, $r_\emptyset$ denotes the empty run).
- If $\delta((r_1, r_2), (u_1, u_2)) = (r'_1, r'_2)$ then $r_1 + u_1 = r'_1$ and $r_2 + u_2 = r'_2$.

A *synchronous step transition system* for N_1 and N_2 is a joint step transition system $TS = (Q, \delta, q_{in})$ for N_1 and N_2 that satisfies the following conditions:

- TS is coherent.
- If $(r_1, r_2) \in Q$ and there is a run $r'_1 \in \mathit{Runs}(N_1)$ and a subset $u_1 \subseteq T_1$ such that $r'_1 = r_1 + u_1$, then there exists $r'_2 \in \mathit{Runs}(N_2)$ and $u_2 \subseteq T_2$ such that $\delta((r_1, r_2), (u_1, u_2)) = (r'_1, r'_2)$.
- If $(r_1, r_2) \in Q$ and there is a run $r'_2 \in \mathit{Runs}(N_2)$ and a subset $u_2 \subseteq T_2$ such that $r'_2 = r_2 + u_2$, then there exists $r'_1 \in \mathit{Runs}(N_1)$ and $u_1 \subseteq T_1$ such that $\delta((r_1, r_2), (u_1, u_2)) = (r'_1, r'_2)$.

Intuitively, a synchronous step transition system for N_1 and N_2 corresponds to a bisimulation between the unfoldings of the two nets in which each step in one net is simulated uniformly by a step from the other net.

Lemma 3. *Let $N_i = (S_i, T_i, F_i, M^i_{in}, \lambda_i)$, $i \in \{1,2\}$ be a pair of trace-labelled Petri nets. There is hereditary history preserving bisimulation between N_1 and N_2 if and only if there is a synchronous step transition system for N_1 and N_2.*

Proof. ($\Rightarrow$) Let H be a hereditary history preserving bisimulation between N_1 and N_2. We construct a synchronous step transition system $TS = (Q, \delta, q_{in})$ for N_1 and N_2 as follows:

- $Q = \{(\rho_1, \rho_2) \mid \rho \in H\}$.
- $q_{in} = (r_\emptyset, r_\emptyset)$.
- $\delta((\rho_1, \rho_2), (u_1, u_2)) = (\rho'_1, \rho'_2)$ if $\rho, \rho' \in H$ with $\rho + \langle u_1, u_2 \rangle = \rho'$.

We first verify that TS is coherent. Suppose that $\delta((\rho_1, \rho_2), (t'_1, t'_2)) = (\rho'_1, \rho'_2)$ and $\delta((\rho'_1, \rho'_2), (t''_1, t''_2)) = (\rho''_1, \rho''_2)$ such that $t'_1 I_1 t''_1$ and $t'_2 I_2 t''_2$. Then, $\rho'' = \rho + \langle \{t'_1, t''_1\}, \{t'_2, t''_2\} \rangle$ so, by the definition of δ, $\delta((\rho_1, \rho_2), (\{t'_1, t''_1\}, \{t'_2, t''_2\})) = (\rho''_1, \rho''_2)$, as required by coherence.

Next, we check forward extensibility. Suppose that (ρ_1, ρ_2) in TS and $\rho_1 + u_1 = \rho'_1 \in \mathit{Runs}(TS)$. Let $u_1 = \{t_1, t_2, \ldots, t_k\}$. Since $\rho \in H$, there must exist transitions $t'_1, t'_2, \ldots, t'_k \in T_2$ such that $\rho + \langle t_1, t'_1 \rangle + \langle t_2, t'_2 \rangle + \cdots + \langle t_k, t'_k \rangle = \rho'' \in H$. Setting $u_2 = \{t'_1, t'_2, \ldots, t'_k\}$, it follows that $\delta((\rho_1, \rho_2), (u_1, u_2)) = (\rho''_1, \rho''_2)$. Clearly, $\rho''_1 = \rho'_1$, so the required transition is present in δ.

($\Leftarrow$) Conversely, let $TS = (Q, \delta, q_{in})$ be a synchronous step transition for N_1 and N_2. Without loss of generality, we assume that every state $(r_1, r_2) \in Q$ is reachable from the initial state $(r_\emptyset, r_\emptyset)$ – if there is an unreachable state, we can remove it from TS without affecting either the coherence or the bisimulation characteristics of TS.

Let $H = \{\rho \mid (\rho_1, \rho_2) \in Q\}$. We claim that H is a hereditary history preserving bisimulation. It is easy to verify that H satisfies the two forward extensibility conditions for hereditary history preserving bisimulations.

What we need to establish is the backward closure condition – if $\rho \in H$ and e is a maximal event in ρ, we must argue that $\rho - e$ also belongs to H. The state (ρ_1, ρ_2) in TS corresponding to ρ is reachable from $(r_\emptyset, r_\emptyset)$. Let $w = (t_1^1, t_1^2), (t_1^2, t_2^2), \ldots, (t_1^k, t_2^k)$ be a sequence of transitions such that $\delta((r_\emptyset, r_\emptyset), w) = (\rho_1, \rho_2)$. The maximal event e in ρ corresponds to some pair of transitions (t_1^j, t_2^j) in this sequence. Since TS is coherent, by Proposition 2, for every reordering w' of w that is consistent with $I_{N_{12}}$, $\delta((r_\emptyset, r_\emptyset), w') = (\rho_1, \rho_2)$. Since e is a maximal event in ρ, there is a reordering of w of the form $w' = u \cdot (t_1^j, t_2^j)$. Let $(r_1^u, r_2^u) = \delta((r_\emptyset, r_\emptyset), u)$. Then, $\delta((r_1^u, r_2^u), (t_1^j, t_2^j)) = (\rho_1, \rho_2)$. It follows, therefore, that the matched run $\rho^u \in H$ such that $\rho_1^u = r_1^u$ and $\rho_2^u = r_2^u$ is the matched run obtained by removing e from ρ. □

6 Decidability of Strong Step Bisimulation

Recall that our goal is to show that hereditary history preserving bisimulation is decidable for 1-safe trace-labelled nets. From the previous section, it suffices to check whether a pair of trace-labelled nets admits a synchronous step transition system. It turns out that this amounts to checking for the existence of a state-based step bisimulation between a pair of finite step transition systems.

Trace-relabelling. Let (A, I_A) and (B, I_B) be a pair of trace alphabets. A trace-relabelling of (A, I_A) by (B, I_B) is a function $\lambda : A \to B$ such that $a I_A a'$ if and only if $\lambda(a) I_B \lambda(a')$.

Step bisimulations. For $i \in \{1, 2\}$, let $TS_i = (Q_i, \delta_i, q_{in}^i)$ be a step transition system over trace alphabet (A_i, I_i), and let $\lambda_i : A_i \to B$ be a trace-relabelling of (A_i, I_i) by a third trace alphabet (B, I_B). A *step bisimulation* between TS_1 and TS_2 is a relation $R \subseteq Q_1 \times Q_2$ such that:

(i) $(q^1_{in}, q^2_{in}) \in R$.
(ii) $(q_1, q_2) \in R$ and $\delta_1(q_1, u_1) = q'_1$ implies there exists u_2 such that $\lambda_1(u_1) = \lambda_2(u_2)$ and $(q'_1, \delta_2(q_2, u_2)) \in R$.
(iii) $(q_1, q_2) \in R$ and $\delta_2(q_2, u_2) = q'_2$ implies there exists u_1 such that $\lambda_1(u_1) = \lambda_2(u_2)$ and $(\delta_1(q_1, u_1), q'_2) \in R$.
(iv) Let $(q_1, q_2) \in R$ and $(\delta_1(q_1, u_1), \delta_2(q_2, u_2)) \in R$ for u_1, u_2 such that $\lambda_1(u_1) = \lambda_2(u_2) = u$. For each $v \subseteq u$, $(\delta(q_1, \lambda_1^{-1}(v)), \delta(q_2, \lambda_2^{-1}(v))) \in R$.

Thus, a step bisimulation is just a bisimulation with respect to an additional level of labelling that respects substeps. Conditions (ii) and (iii) are the normal forward extensibility criteria for bisimulation, extended to steps. The fourth condition ensures that R is closed under substeps.

Run foldings. Let (Σ, I) be a trace alphabet and let $N = (S, T, F, M_{in}, \lambda)$ be a trace-labelled net over Σ. For each run $r \in \mathit{Runs}(N)$, recall that M_r denotes the marking associated with r. Let $\mathit{top}_r \subseteq \Sigma$ denote the labels associated with the maximal transitions in r. Clearly, top_r is a subset of Σ in which all actions are pairwise independent. The *run folding* of N is the step transition system $TS = (Q, \delta, q_{in})$ over (T, I_N) where:

- $Q = \{(M_r, \mathit{top}_r) \mid r \in \mathit{Runs}(N)\}$.
- $q_{in} = (M_{in}, \emptyset)$.
- For all $u \subseteq T$ such that $M_r \xrightarrow{u}$, $\delta((M_r, \mathit{top}_r), u) = (M_{r+u}, \mathit{top}_{r+u})$.

Observe that the run folding of N is a finite step transition system. The transition function δ is well-defined because N is trace-labelled – it is not difficult to see that for any pair of runs r, r' such that $M_r = M_{r'}$, $\mathit{top}_r = \mathit{top}_{r'}$ and $M_r \xrightarrow{u}$, $\mathit{top}_{r+u} = \mathit{top}_{r'+u}$.

Lemma 4. *Let (Σ, I) be a trace alphabet and $N_i = (S_i, T_i, F_i, M^i_{in}, \lambda_i)$, $i = \{1, 2\}$ be a pair of trace-labelled nets over Σ. Then, N_1 and N_2 admit a synchronized step transition system if and only if there exists a step bisimulation between the run foldings of N_1 and N_2 with trace-relabellings λ_1 and λ_2 from T_1 and T_2 into Σ, respectively.*

Proof. $(\Rightarrow)$ Let $TS = (Q, \delta, q_{in})$ be a synchronized step transition system for N_1 and N_2 and let $TS_1 = (Q_1, \delta_1, q^1_{in})$ and $TS_2 = (Q_2, \delta_2, q^2_{in})$ be the run foldings of N_1 and N_2, respectively. Recall that each state in TS is of the form (r_1, r_2), where $r_1 \in \mathit{Runs}(N_1)$ and $r_2 \in \mathit{Runs}(N_2)$ and each state in TS_i, $i \in \{1, 2\}$, is of the form (M_r, top_r) for some run $r \in \mathit{Runs}(N_i)$.

We define a relation $R \subseteq Q_1 \times Q_2$ as follows:
$R = \{((M_1, A_1), (M_2, A_2)) \mid (r_1, r_2) \in Q, M_i = M_{r_i}, A_i = \mathit{top}_{r_i}, i \in \{1, 2\}\}$

The fact that R is a step bisimulation between TS_1 and TS_2 follows immediately from the definition of a synchronized step transition system for N_1 and N_2.

The forward extensibility conditions for R follow from the extensibility criteria for TS. The substep closure of R follows from the coherence of TS.

($\Leftarrow$) Conversely, suppose that R is a step bisimulation between TS_1 and TS_2. We have to construct a synchronized step transition system TS for N_1 and N_2.

We construct TS inductively, maintaining the invariant that for every state (r_1, r_2) that we add to TS, the corresponding pair of states $((M_{r_1}, top_{r_1}), (M_{r_2}, top_{r_2}))$ belongs to R.

We begin with the initial state $(r_\emptyset, r_\emptyset)$ that corresponds to the pair $((M^1_{in}, \emptyset), (M^2_{in}, \emptyset))$ which is guaranteed to belong to R.

Let (r_1, r_2) be a state in TS. We know that $((M_{r_1}, top_{r_1}), (M_{r_2}, top_{r_2})) \in R$. For each pair (u_1, u_2) such that $(\delta_1((M_{r_1}, top_{r_1}), u_1), \delta_2((M_{r_2}, top_{r_2}), u_2)) \in R$, add the state $(r_1 + u_1, r_2 + u_2)$ to TS.

It is easy to see that TS is a joint step transition system for N_1 and N_2. To check that it is, in fact, a synchronized step transition system for N_1 and N_2, we must establish coherence and the forward extensibility properties.

We first check coherence. Suppose that $\delta((r_1, r_2), (t_1, t_2)) = (r'_1, r'_2)$ and $\delta((r'_1, r'_2), (t'_1, t'_2)) = (r''_1, r''_2)$ where $(t_1, t_2) I_{N_{12}} (t'_1, t'_2)$. We need to show that $\delta((r_1, r_2), \{(t_1, t_2), (t'_1, t'_2)\}) = (r''_1, r''_2)$ as well. From the construction of TS, we know that both $((M_{r_1}, top_{r_1}), (M_{r_2}, top_{r_2}))$ and $((M_{r''_1}, top_{r''_1}), (M_{r''_2}, top_{r''_2}))$ belong to R. From the definition of run foldings, we have $\delta_i((M_{r_i}, top_{r_i}), \{t_i, t'_i\}) = (M_{r''_i}, top_{r''_i})$ for $i \in \{1, 2\}$. Hence, $\delta((r_1, r_2), \{(t_1, t_2), (t'_1, t'_2)\}) = (r''_1, r''_2)$.

The forward extensibility properties for TS follow naturally from the properties of step bisimulations, so we omit the details. □

Theorem 5. *Hereditary history preserving bisimulation is decidable for trace-labelled systems.*

Proof. From Lemmas 3 and 4, it follows that a pair of nets N_1 and N_2 admit a hereditary history preserving bisimulation if and only if there is a step bisimulation between their run foldings. It is clear that the existence of a step bisimulation between two finite step transition systems can be checked (exhaustively, if nothing else). Since the run foldings of N_1 and N_2 are guaranteed to be finite step transition systems, the result follows. □

7 Discussion

As we mentioned at the outset, while branching time semantics is relatively well understood in the interleaved approach to the semantics of concurrent systems, the situation is not so clear when labelled partial orders are used to record the behaviour of such systems. In many contexts, the application of Mazurkiewicz trace theory provides a context in which apparently intractable problems become solvable. Our result here is one example of this phenomenon. This seems to suggest that it might make sense to restrict our attention to trace-labelled systems, at least initially, when attempting to build up our understanding of the interplay between nondeterminism and concurrency.

References

1. M.A. Bednarczyk: Hereditary history preserving bisimulation or what is the power of the future perfect in program logics. Technical report, Polish Academy of Sciences, Gdansk (1991).
2. M.A. Bednarczyk: Categories of asynchronous systems, PhD Thesis, Report 1/88, Computer Science, University of Sussex (1988).
3. P. Degano, R. de Nicola and U. Montanari: Partial ordering descriptions and observations of nondeterministic concurrent processes, in *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, Springer LNCS **354** (1989) 438–466.
4. V. Diekert and G. Rozenberg (eds), *The Book of Traces*, World Scientific, Singapore (1995).
5. S.B. Fröschle and T.T. Hildebrandt: On plain and hereditary history-preserving bisimulation, *Proc MFCS '99*, Springer LNCS **1672** (1999) 354–365.
6. L. Jategaonkar and A. Meyer: Deciding true concurrency equivalence on safe, finite nets, *Theoretical Computer Science*, **154**(1) (1996) 107–143.
7. A. Joyal, M. Nielsen and G. Winskel: Bisimulation from open maps, *Information and Computation*, **127**(2) (1996) 164–185.
8. M. Jurdzinski and M. Nielsen: Hereditary history preserving bisimilarity is undecidable, in *Proc. STACS 2000* Springer LNCS **1770** (2000) 358–369.
9. K. Lodaya, R. Parikh, R. Ramanujam and P.S. Thiagarajan: A Logical Study of Distributed Transition Systems, *Information and Computation* **119**(1) (1995) 91–118.
10. A. Mazurkiewicz: Basic notions of trace theory, in: J.W. de Bakker, W.-P. de Roever, G. Rozenberg (eds.), *Linear time, branching time and partial order in logics and models for concurrency*, *LNCS*, **354** (1989) 285–363.
11. R. Milner: *Communication and Concurrency*, Prentice-Hall, London (1989).
12. M. Nielsen and C. Clausen: Games and Logics for a Noninterleaving Bisimulation, *Nordic Journal of Computing* **2**(2) (1995) 221–249.
13. D. Park: Concurrency and automata on infinite sequences, in *Proc. Theoretical Computer Science: 5th GI Conference*, Springer LNCS **104** (1981).
14. W. Reisig and G. Rozenberg (eds.): *Lectures on Petri Nets, Vols I and II Springer LNCS* **1492,1493** (1998).
15. A. Rabinovitch and B.A. Trakhtenbrot: Behavior strucures and nets, *Fundamenta Informaticae*, **11**(4) (1988) 357–403.
16. R. van Glabbeek and U. Goltz: Refinement of actions and equivalence notions for concurrent systems, *Proc MFCS '89*, Springer LNCS **379** (1989) 237–248.
17. G. Winskel and M. Nielsen: Models for concurrency, in S. Abramsky, D. Gabbay and T.S.E. Maibaum, eds, *Handbook of Logic in Computer Science*, *Vol 4*, Oxford (1995) 1–148.
18. W. Zielonka: Notes on finite asynchronous automata, *R.A.I.R.O.–Inform. Théor. Appl.*, **21** (1987) 99–135.

Lower Bounds for Embedding Graphs into Graphs of Smaller Characteristic

K.V.M. Naidu and H. Ramesh

Indian Institute of Science, Bangalore, 560012
ramesh@csa.iisc.ernet.in, naidukvm@yahoo.com

Abstract. The subject of *graph embeddings* deals with embedding a finite point set in a given metric space by points in another target metric space in such a way that distances in the new space are at least, but not too much more, than distances in the old space. The largest new distance to old distance ratio over all pairs of points is called the *distortion* of the embedding. In this paper, we will study the distortion $dist(G, H)$ while embedding metrics supported on a given graph G into metrics supported on a graph H of lower *characteristic*, where the characteristic $\chi(H)$ of a graph H is the quantity $E - V + 1$ (E is the number of edges and V is the number of vertices in H). We will prove the following lower bounds for such embeddings which generalize and improve lower bounds given in [10].

– If $|G| = |H|$ and $\chi(G) - \chi(H) = k$, $dist(G, H) \geq g_k - 1$
– If $\chi(G) - \chi(H) = k$, $dist(G, H) \geq \frac{g_k - 4}{3}$

Further, we will also give an alternative proof for lower bounding the distortion when probabilistically embedding expander graphs into tree metrics. In addition, we also generalize this lower bound to the case when expander graphs probabilistically embed into graphs of constant characteristic.

1 Introduction

The subject of graph embeddings deals with representing a finite point set in a metric space by points in another target metric space in such a way that the distances only increase, but not by too much. Once a given metric space can be approximately embedded into a metric space that is easier to deal with, one might get approximate solutions to problems which appeared to be hard to deal with in the original metric space. For a nice introduction to to this area refer to [7], [6] and [1].

This approach has been successfully adopted in several problems. One example would be for clustering points: here, one could embed the points at hand into some metric space such as the ℓ_2-space and try geometric algorithms that are already well known. Another example is bandwidth approximation. The first non-trivial approximation algorithm for bandwidth approximation was given in [9] using a generalization of the embeddings which form the subject of this paper. Bartal in [2] gave an algorithm to probabilistically approximate arbitrary graph metrics by tree metrics within a distortion factor of $O(\log^2(n))$, and subsequently, improved it in [3] to a factor of $O(\log n \log\log n)$. This lead to the design of polylog approximation algorithms for several problems. See [5] and [8] for examples of such problems. Charikar et al. in [4] showed how to derandomize approximation algorithms designed using probabilistic approximation by trees.

We define the *characteristic* $\chi(G)$ of a graph G as the value $E - V + 1$, where E, V are the number of edges and vertices, respectively, in G. In this paper, we examine the distortion obtained when embedding metric spaces supported on graphs into metric spaces supported on graphs of lower characteristic, both deterministically and probabilistically. Our results are stated as follows:

Previous Results: Rabinovich and Raz in [10] proved the following results on embedding a graph G into a graph H of lower characteristic. Let G and H be unweighted graphs, where $|G|$ denotes the number of vertices of G, $\chi(G)$ denotes the characteristic of G, g_k denotes the length of the kth shortest cycle in G, and $dist(G, H)$ denotes the maximum distortion over all edges resulting from the best embedding of G into H.

– If $|G| = |H|$ and $\chi(G) > \chi(H)$, $dist(G, H) \geq \frac{g_1}{3} - 1$
– If $\chi(G) > \chi(H)$, $dist(G, H) \geq \frac{g_1}{4} - \frac{1}{2}$
– If $\chi(G) - 1 > \chi(H)$, $dist(G, H) \geq \frac{g_2}{4} - \frac{3}{2}$

Whether the lower bound even in case of $|G| \neq |H|$ is $\frac{g_1}{3} - 1$ is left as a open question in [10]. Also left as open is whether one can generalize the last of the statements above.

Our Results: We prove the following results.

– If $|G| = |H|$ and $\chi(G) - \chi(H) = k$, $dist(G, H) \geq g_k - 1$
– If $\chi(G) - \chi(H) = k$, $dist(G, H) \geq \frac{g_k - 4}{3}$

The first result both generalizes and strengthens the first result in [10] stated above. The second result generalizes the second and third results in [10] and is also stronger for not too small values of g_k. Our results are obtained using topological arguments which relate cycles and linear combinations of cycles in G to corresponding structures in H, and vice versa.

2 Preliminaries

In this section we give some basic definitions that will be useful throughout the paper.

Definition 1. For any set N and a function $\rho : N \times N \to \Re$, we call (N, ρ), a *metric space* if
• $\forall x, y \in N \quad \rho(x, y) \geq 0$ with equality holding iff $x = y$
• $\forall x, y \in N \quad \rho(x, y) = \rho(y, x)$
• $\forall x, y, z \in N \quad \rho(x, y) + \rho(y, z) \geq \rho(x, z)$

Definition 2. Let (N, ρ) and (M, σ) be metric spaces and $g : N \to M$ be a function, such that:

$$\forall x, y \in N \quad \rho(x, y) \leq \sigma(g(x), g(y)) \leq D\rho(x, y)$$

The infimum of all such numbers D is called the distortion of g and is denoted by $D(g)$. In this paper, we will consider only functions g which are *expansive*, i.e., $\frac{\sigma(g(x),g(y))}{\rho(x,y)} \geq 1$, for all $x, y \in N$, $x \neq y$.

Graphs and Metric Spaces. With any edge-weighted graph $G = (V, E, W)$, one can naturally associate a metric space by defining the distance between any two vertices as the length of the shortest path between them. We call such a metric space as a metric *supported* on G. Conversely, any finite metric space can be viewed as a weighted graph. Just take a complete graph and weigh each edge with the distance between the corresponding points.

Let G be a weighted graph and (N, ρ) be a metric space. The distortion of the best embedding of G into N is denoted by $dist(G, N)$. Alternately, it is the infimum of $D(g)$, over all expansive functions g from G to N. Let S be a set of metric spaces. $dist(G, S)$ is similarly defined as infimum of $dist(G, N)$ over all metric spaces $N \in S$.

We say that a metric space (N, ρ) is *dominated* by a metric space (M, σ) through a function $g : N \to M$ if $\forall x, y \in N \quad \rho(x, y) \leq \sigma(g(x), g(y))$.

Probabilistic Embeddings. We also consider probabilistic embeddings in this paper. A graph G is said to be α*-probabilistically* approximated by a finite set of metric spaces S, if both the following conditions hold:

- There is a probability assignment to elements (M, σ) of S such that each element with non-zero associated probability dominates G through an associated function $g_{(M,\sigma)}$.
- For every pair of vertices x, y in G, the expected distance

  $$Expectation_{(M,\sigma)} \left(\sigma(g_{(M,\sigma)}(x), g_{(M,\sigma)}(y))\right)$$

  is at most α times their distance in G.

We also denote the infimum of all such αs as $pdist(G, S)$, where the infimum is taken over all probability distributions.

Going from Discrete to Continuous. Now, we will introduce a notion which allows us to work with continuous counterparts of discrete objects like graphs and edges. This notion was introduced in [10] to prove similar lower bounds. Consider any edge (from an unweighted graph) and associate with it a unit interval with the line metric (for weighted graphs, one would use intervals of length equal to the edge weight). Now, instead of unweighted graphs we work with vertices and unit intervals connecting them, in the sense that end points of the interval are identified with the respective end points of the edges. Such a continuous structure obtained from a graph G will be called $\tilde{G}$. Note that there is a metric space naturally associated with this continuous structure. The distance between any two points (where a point could be a vertex or internal to an edge) in $\tilde{G}$ is defined as the length of the shortest path connecting them. Such paths are also called *geodesic paths*. Also, note that the distance between any two vertices in $\tilde{G}$ is the same as the distance between these vertices in G.

We can also extend any 1-1 mapping, g, from the vertices of a graph G into the vertices of a graph H, to get a continuous, many-one mapping $\tilde{g}$ from $\tilde{G}$ into $\tilde{H}$, whose restriction to the vertex set of G gives us back the original mapping g. $\tilde{g}$ is called the *continuous extension* of g, and can be obtained in several ways. We obtain $\tilde{g}$ from g using a linearly defined extension, described below.

Consider any point v in $\tilde{G}$; we show how to define $\tilde{g}(v)$. If v is a vertex in G, then $\tilde{g}(v) = g(v)$. So consider the case when v is not a vertex of G. Let (u, w) be the edge in $\tilde{G}$ containing v. There could be several geodesic paths in $\tilde{H}$ that connect $g(u)$ to $g(w)$ in $\tilde{H}$. Before defining the extension, we choose any one of these geodesic paths. Intuitively, we linearly map points on the edge (u, w) in $\tilde{G}$ to this geodesic path in $\tilde{H}$. More formally, define $d_{\tilde{G}}(u, v)$ to be the distance between u and v in $\tilde{G}$, and let:

$$\alpha = \frac{d_{\tilde{G}}(u, v)}{d_{\tilde{G}}(u, w)}.$$

Define $\tilde{g}(v)$ as the unique point y on the above chosen geodesic path associated with (u, w) such that

$$\frac{d_{\tilde{H}}(g(u), y)}{d_{\tilde{H}}(g(u), g(w))} = \alpha.$$

Using the above map, [10] gave a proof for the fact that any cycle of length n would have a distortion of $\Omega(n)$ when approximated by trees.

Lemma 1 was originally proved in [10] and will be useful in our proofs.

Lemma 1. *[10] If there exists $x, y \in \tilde{G}$ such that $d_{\tilde{G}}(x, y) \geq d$ and $\tilde{g}(x) = \tilde{g}(y)$, then $dist(G, H) \geq d - 1$. Note that x, y need not be vertices of G. They are points on the continuous extension of G.*

3 Lower Bounds on Distortion

In this section, we prove a range of lower bounds on approximating a graph metric supported on a given unweighted graph G by a metric supported on a graph (possibly weighted with each edge length being at least 1) H of lower characteristic. We also prove a lower bound on approximating expander graphs by a distribution on metrics supported on graphs of lower characteristic.

3.1 Intuition

Suppose we try to show that approximating a metric supported on a cycle $\mathcal{C}$ (of say 4 vertices, $ABCDA$, in that order) by a metric supported on a tree $\mathcal{T}$ (with four vertices), incurs a distortion of at least $|\mathcal{C}| - 1$. The proof would proceed as follows.

Consider any 1-1 map g [1] from the vertices of $\mathcal{C}$ to the vertices of $\mathcal{T}$, and let $\tilde{g}$ be its continuous extension. Consider traversing $\mathcal{C}$ in the order $ABCDA$, and consider the image (under $\tilde{g}$) of this trajectory. The resulting image trajectory is a closed walk on $\mathcal{T}$ which is *self-cancelling*, i.e., each edge is traversed twice, once each in opposite directions. Next consider the image of this self-cancelling closed walk under the map $\widetilde{g^{-1}}$. Note that this image is also a self-cancelling closed walk on $\mathcal{C}$. Thus, under the map $\tilde{g} \circ \widetilde{g^{-1}}$, the circle $\mathcal{C}$ maps to a self-cancelling closed walk on $\mathcal{C}$.

Note that vertices in $\mathcal{C}$ map to themselves under $\tilde{g} \circ \widetilde{g^{-1}}$. This along with the fact that the circle $\mathcal{C}$ maps to a self-cancelling closed walk on $\mathcal{C}$ under $\tilde{g} \circ \widetilde{g^{-1}}$ implies that there exists at least one cycle edge which has a distortion of $|\mathcal{C}| - 1$ under $\tilde{g} \circ \widetilde{g^{-1}}$. Since g is expansive, the distortion under $\tilde{g}$ is at least the distortion under $\tilde{g} \circ \widetilde{g^{-1}}$. It follows that some edge of $\mathcal{C}$ has distortion at least $|\mathcal{C}| - 1$ under $\tilde{g}$ and therefore under g.

3.2 Generalization

We now make the above arguments more general. Throughout the section, we use $\chi(G)$ to denote the characteristic of a graph G, which is defined as $|E_G| - |V_G| + 1$. $\chi(G)$ is the number of independent cycles in a graph in the sense that all cycles in G can be expressed in terms of linear combinations of a *cycle basis* having $\chi(G)$ cycles in it.

1. Assume that a map g is given from V_G to V_H. We will use the continuous extension $\tilde{G}$ of G and $\tilde{g}$ of g as defined in Section 2. Let $h = g^{-1}$ and consider the maps, $\tilde{G} \xrightarrow{\tilde{g}} \tilde{H} \xrightarrow{\tilde{h}} \tilde{G}$.
2. Clearly, each cycle in the cycle basis of $\tilde{G}$ maps to a cycle in $\tilde{H}$ under the map $\tilde{g}$ (actually it will map to a closed walk in $\tilde{H}$ but cancelling out edges traversed in opposite directions will leave a cycle; this cycle could also be trivial, if all edges end up cancelling out). The total number of independent non-trivial cycles that could be obtained is clearly at most $\chi(H)$.
3. Next, consider each of the non-trivial cycles obtained above and map them back to $\tilde{G}$ using the map $\tilde{h}$. Again, we obtain a new collection of cycles in $\tilde{G}$ (by cancelling out edges traversed in opposite directions). Let *dim* denote the number of independent non-trivial cycles obtained in $\tilde{G}$. Clearly, *dim* is at most $\min\{\chi(G), \chi(H)\} = \chi(H)$ (because we started out with exactly $\chi(G)$ cycles in $\tilde{G}$ and got at most $\chi(H)$ cycles in the previous step).
4. Consider each cycle C in the cycle basis of $\tilde{G}$ and consider the cycle obtained by applying the map $\tilde{f} = \tilde{g} \circ \tilde{h}$ to C and cancelling out portions traversed in opposite directions. We claim that this cycle is exactly the same as that obtained by Steps 2 and 3 above for the following reason: any part of the

[1] Any expansive map has to be 1-1, as two vertices in $\mathcal{C}$ cannot get mapped to the same vertex in $\mathcal{T}$

cycle that cancels out under the map $\tilde{g}$ will cancel out under the map $\tilde{f}$ as well. It follows that the number of independent non-trivial cycles obtained by applying $\tilde{f}$ to the cycles in the above basis is dim as well.
5. We then show that if the distortion under the map $\tilde{g} \circ \tilde{h}$ is small, then $dim > \chi(H)$, which is a contradiction. It follows that the distortion under the map $\tilde{g} \circ \tilde{h}$ must be large.
6. Finally, the proof is completed by the fact that the distortion under $\tilde{g}$ is at least the distortion under $\tilde{g} \circ \tilde{h}$, since g is expansive.

All the steps above except Step 5 are ready. We concentrate on Step 5 for proving the lower bounds in the rest of the paper. All the lower bound proofs will conclude with a contradiction showing that $dim > \chi(H)$, as required by Step 5.

Special Cycle Basis. Some of the proofs require a special kind of cycle basis for G constructed as follows.

- Process cycles in G in increasing order of length.
- Include a cycle in the basis if and only if it can not be expressed as a linear combination of cycles occurring before it.

By the definition of $\chi(G)$, the size of the basis is $\chi(G)$. A key property of the above basis is that, if the ith smallest cycle is in the basis, then all cycles smaller than it are expressible as linear combinations of basis cycles smaller than the ith cycle. This property is important in the proofs.

3.3 Lower Bounds

In this section, we will prove lower bounds on approximating a graph metric by a metric supported on a graph of lower characteristic. We use $|P|$ to denote the length of the path P. Also, we do additions of paths algebraically with the same edge cancelling out if traversed in opposite directions. Note that any vertex of G maps to itself under the map $\tilde{f} = \tilde{g} \circ \tilde{h}$ where $\tilde{g}$ is expansive and $h = g^{-1}$.

Theorem 1. *If $|G| = |H|$ and $\chi(H) < \chi(G)$, then $dist(G, H) \geq g_1 - 1$, where g_1 is length of the shortest cycle in G.*

Proof: Let g denote any expansive map from the vertices of G to that of H and suppose the distortion under g is less than $g_1 - 1$. We show a contradiction as follows.

Consider any cycle basis B of G. Consider any cycle $C_i \in B$. For each edge e_i^j in C_i,

$$|e_i^j - \tilde{f}(e_i^j)| \leq |e_i^j| + |\tilde{f}(e_i^j)| \leq |e_i^j| + |\tilde{g}(e_i^j)| < 1 + (g_1 - 1).1 = g_1$$

The second inequality follows from the fact that g is expansive, and therefore distances between vertices of H can only decrease under the map $\tilde{g^{-1}}$. Next, note that a consequence of the above is that e_i^j and $-\tilde{f}(e_i^j)$ together cannot form a cycle in G as the length of any cycle must be at least g_1. Since $e_i^j - \tilde{f}(e_i^j)$ is a closed walk, $e_i^j - \tilde{f}(e_i^j) = 0$ or simply, $\tilde{f}(e_i^j) = e_i^j$.

It follows that each cycle C_i in the basis being considered maps to itself under the map $\tilde{f}$, and therefore $dim = \chi(G) > \chi(H)$, as required. □

Theorem 2. *If $|G| = |H|$ and $\chi(G) - \chi(H) = k$, then $dist(G, H) \geq g_k - 1$, where g_k is the length of the kth smallest cycle in G.*

Proof: Let g denote any expansive map from the vertices of G to that of H and suppose the distortion under g is less than $g_k - 1$. We show a contradiction as follows. Let S be the special cycle basis which we constructed in the previous subsection. Consider any cycle $C_i \in S$ such that $i \geq k$. Let,

$$\forall i, j \qquad e_i^j - \tilde{f}(e_i^j) = F_i^j$$

By arguing as in Theorem 1 we get,

$$\forall i, j \qquad |F_i^j| < g_k$$

So F_i^j must be either one of the smallest $k-1$ cycles or must completely cancel out. Then, $\sum_j \tilde{f}(e_i^j)$ cannot be written as a linear combination of cycles smaller than C_i, otherwise, since $C_i = \sum_j \tilde{f}(e_i^j) + \sum_j F_i^j$ and since $C_i \neq 0$, C_i becomes a linear combination of smaller cycles, a contradiction. Further, since $\tilde{f}(C_i) = \sum_j \tilde{f}(e_i^j)$ can be written as a linear combination of C_i and cycles smaller than C_i, the cycles $\tilde{f}(C_i)$ are all independent for $i \geq k$. This implies that $dim \geq \chi(G) - k + 1 > \chi(H)$, as required. □

Theorem 3. *If $\chi(G) - \chi(H) = k$, then $dist(G, H) \geq \frac{g_k}{3} - \frac{4}{3}$*

The main bottleneck for proving this theorem seems to be proving that each edge together with it's image under $\tilde{f}$ forms a cycle of small length. The reason why the argument in Theorem 2 does not work directly is because of the presence of extra vertices in H. The distance between the extra vertices need not contract when mapped back to G. This is the reason we get a weaker lower bound compared to the case when the number of vertices in G and H is the same.

Proof of Theorem 3: Let g denote any expansive map from the vertices of G to that of H and suppose the distortion under g is less than $\frac{g_k}{3} - \frac{4}{3}$.

To prove this theorem, we will add additional vertices to G and H. We will continue to call the resulting graphs G and H for convenience. Since the additional vertices will be internal to edges, each addition will increase the number of vertices and number of edges by 1, and therefore, there will be no change in the characteristic of the graphs in question.

First, we will add vertices to H such that distance between any two adjacent vertices is at most 1. This can be done by putting vertices at intervals of 1 on

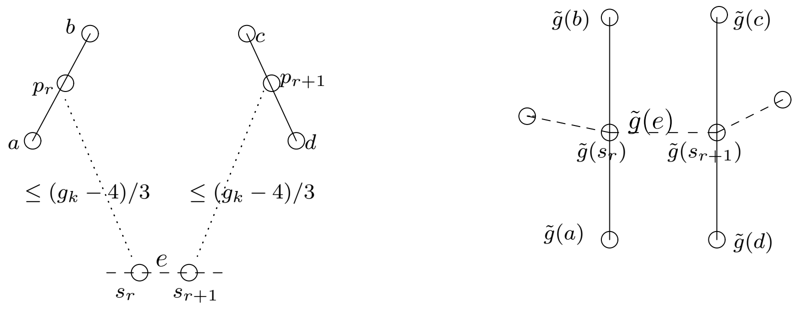


Fig. 1. G on the left and H on the right

each edge. Second, for each vertex v in H to which no vertex in G maps under g, a point on some edge of $\tilde{G}$ that maps to v under the map $\tilde{g}$ is made a vertex in G, provided such a point exists. Note that there could be several such points in $\tilde{G}$, in which case any one is chosen. Finally, the map g from the original vertices of G to the original vertices of H is now extended to include the new vertices of G. Note that the extended map is still 1-1 (i.e., two vertices in G do not map to the same vertex in H). Therefore, $h = g^{-1}$ is well-defined. As before, we consider $\tilde{g}$, $\tilde{h}$, and $\tilde{f} = \tilde{g} \circ \tilde{h}$.

We will assume that any two points in $\tilde{G}$ mapped to the same point in $\tilde{H}$ by $\tilde{g}$ are at a distance at most $\frac{g_k}{3} - \frac{4}{3}$ in $\tilde{G}$, otherwise the proof follows by applying Lemma 1 in Section 2. We will derive a contradiction as follows.

The proof is similar to the proof of Theorem 2, with some key differences. Recall that S denotes the special basis constructed earlier and consider any cycle $C_i \in S$ such that $i \geq k$. Let e_i^j be an edge of this cycle. Mark points $s_1, \ldots, s_l$ on e_i^j that are inverse images of vertices in $\tilde{H}$ which lie on the geodesic $\tilde{g}(e_i^j)$ in $\tilde{H}$. We call the part of the edge between s_r and s_{r+1}, for $r = 1, \ldots, l-1$, an *edgelet*. Instead of working with edges, we will work with edgelets. For the rest of the proof, redefine e_i^j to be the edgelet (s_r, s_{r+1}). Let $p_r = \tilde{f}(s_r)$ and $p_{r+1} = \tilde{f}(s_{r+1})$ be vertices in $\tilde{G}$. See Figure 1.

Next, we would like to show (as in Theorem 2) that the quantity $|e_i^j - \tilde{f}(e_i^j)|$ is strictly less than g_k for each edgelet. However, this alone is not sufficient as p_r need not equal s_r and similarly p_{r+1} need not equal s_{r+1} and consequently, $e_i^j - \tilde{f}(e_i^j)$ need not form a closed walk (in contrast, recall that in earlier theorems, for each vertex v in G, $\tilde{f}(v) = v$). To get around this, we will consider instead, for each edgelet (s_r, s_{r+1}), the walk obtained by starting from s_r, taking the geodesic (in $\tilde{G}$) to p_r, and then to p_{r+1}, and then to s_{r+1}; this walk replaces $\tilde{f}(e_i^j)$ and will be called $\tilde{f}'(e_i^j)$. We will now show that $|e_i^j - \tilde{f}'(e_i^j)| < g_k$. Once this is shown, using the additional fact that the sum of $\tilde{f}'(e_i^j)$ over all edgelets in C_i is identical to the sum of $\tilde{f}(e_i^j)$ over all edgelets in C_i, the rest of the proof is identical to Theorem 2.

To show that $|e_i^j - \tilde{f}'(e_i^j)| < g_k$, it suffices to show that $|\tilde{f}'(e_i^j)| < g_k - 1$. Clearly, $|\tilde{f}'(e_i^j)| \leq d_{\tilde{G}}(s_r, p_r) + d_{\tilde{G}}(p_r, p_{r+1}) + d_{\tilde{G}}(p_{r+1}, s_{r+1})$. Note that $\tilde{g}(s_r) =$

$\tilde{g}(p_r)$, and likewise for s_{r+1}, p_{r+1}. Then, by our assumption above, we know that $d_{\tilde{G}}(s_r, p_r) \leq \frac{g_k}{3} - \frac{4}{3}$ and $d_{\tilde{G}}(s_{r+1}, p_{r+1}) \leq \frac{g_k}{3} - \frac{4}{3}$. It follows that

$$|\tilde{f}'(e_i^j)| \leq d_{\tilde{G}}(p_r, p_{r+1}) + \frac{2g_k}{3} - \frac{8}{3}$$

Next, we bound the first term on the right hand side.

Let $p_r \in$ edge (a, b) and $p_{r+1} \in$ edge (c, d), where $(a, b), (c, d)$ are the original edges in G, i.e., these edges were present even before additional vertices were added at the beginning of this proof. Note that

$$d_{\tilde{H}}(\tilde{g}(a), \tilde{g}(b)) = d_{\tilde{H}}(\tilde{g}(a), \tilde{g}(s_r)) + d_{\tilde{H}}(\tilde{g}(s_r), \tilde{g}(b)) < \frac{g_k}{3} - \frac{4}{3}$$
$$d_{\tilde{H}}(\tilde{g}(c), \tilde{g}(d)) = d_{\tilde{H}}(\tilde{g}(c), \tilde{g}(s_{r+1})) + d_{\tilde{H}}(\tilde{g}(s_{r+1}), \tilde{g}(d)) < \frac{g_k}{3} - \frac{4}{3}$$

because of our assumption of small distortion for the original edges in G. Therefore,

$d_{\tilde{H}}(\tilde{g}(a), \tilde{g}(c)) + d_{\tilde{H}}(\tilde{g}(b), \tilde{g}(d))$
$\leq d_{\tilde{H}}(\tilde{g}(a), \tilde{g}(s_r)) + d_{\tilde{H}}(\tilde{g}(s_r), \tilde{g}(b)) + d_{\tilde{H}}(\tilde{g}(c), \tilde{g}(s_{r+1})) + d_{\tilde{H}}(\tilde{g}(s_{r+1}), \tilde{g}(d)) +$
$2d_{\tilde{H}}(\tilde{g}(s_r), \tilde{g}(s_{r+1}))$
$< \frac{2g_k}{3} - \frac{8}{3} + 2 = \frac{2g_k}{3} - \frac{2}{3}$

The second inequality above is due to the fact that the distance between consecutive vertices in $\tilde{H}$ is at most 1, by construction. Next, assuming wlog that $d_{\tilde{H}}(\tilde{g}(a), \tilde{g}(c)) \leq d_{\tilde{H}}(\tilde{g}(b), \tilde{g}(d))$ and using the fact that $\tilde{g}$ is expansive on the original edges of G, we get: $d_{\tilde{G}}(a, c) \leq d_{\tilde{H}}(\tilde{g}(a), \tilde{g}(c)) < \frac{g_k}{3} - \frac{1}{3}$. Since $d_{\tilde{G}}(p_r, p_{r+1}) \leq d_{\tilde{G}}(a, c) + 2 < \frac{g_k}{3} - \frac{1}{3} + 2$, we get that

$$|\tilde{f}'(e_i^j)| < \frac{g_k}{3} - \frac{1}{3} + 2 + \frac{2g_k}{3} - \frac{8}{3} = g_k - 1$$

and therefore $|e_i^j - \tilde{f}'(e_i^j)|$ is strictly less than g_k, as required. □

Next, we use Yao's Lemma (stated below) to prove probabilistic lower bounds on embedding expander graphs in graphs of constant characteristic.

Theorem 4. *Let us denote by $\{p\}$, a probability distribution over metric spaces from S and by $\{q\}$, a probability distribution over $(u, v) \in E_G$. Then*

$$\min_{\{p\}} \max_{(u,v) \in E_G} \sum_{H \in S} p_H \frac{d_H(u, v)}{d_G(u, v)} \geq \max_{\{q\}} \min_{H \in S} \sum_{(u,v) \in E_G} q_{uv} \frac{d_H(u, v)}{d_G(u, v)} \qquad (1)$$

Theorem 5. *Let G be a graph of n vertices, m edges and girth g_1, and let S be a family of graphs with characteristic at most $\chi(G) - k(n)$. Consider embeddings of G into each of the graphs in S. For any probability distribution over S, there must be an edge in G whose expected distortion under the above embeddings is at least $\frac{k(n)(g_1 - 7)}{3E_G} + 1$.*

Proof: We use Theorem 4 with uniform distribution over the edges of G. Fix any $H \in S$ and consider the map g embedding G in H. We show that there must be at least $k(n)$ edges having a distortion at least $\frac{g_1-4}{3}$ under g. This being the case, the average distortion over all edges is, as required, at least:

$$\frac{1}{|E_G|}[k(n)\frac{g_1-4}{3} + (E_G - k(n))1] \geq \frac{k(n)(g_1-7)}{3E_G} + 1$$

For a contradiction, suppose there are $l < k(n)$ edges having distortion at least $\frac{g_1-4}{3}$. Remove these edges and consider the remaining graph G'. The characteristic of this graph is $\chi(G) - l > \chi(G) - k(n) \geq \chi(H)$ and its girth is at least g_1. Every edge in G', and consequently, every pair of vertices in G' has distortion less than $\frac{g_1-4}{3}$ in the embedding of G' in H. But by Theorem 3, since $\chi(G') > \chi(H)$, there must be a pair of vertices in G' with distortion at least $\frac{g_1-4}{3}$, a contradiction. □

Corollary 31 *An Expander graph with degree c and girth $g_1(c)\log n$ cannot be probabilistically approximated strictly within a factor of $\frac{k(n)(g_1(c)\log n-7)}{1.5cn} + 1$ by metrics over graphs having characteristic lower by at least $k(n)$. In particular, it cannot be probabilistically approximated within an $\Omega(\log n)$ factor, by graphs of constant characteristic.*

References

1. Anupam Gupta: *Embeddings of Finite Metrics*, Ph.D. Thesis, Computer Science, University of California, Berkeley, 2000.
2. Y. Bartal: *Probabilistic Approximation of Metric Spaces and its Algorithmic Applications*, Proceedings of the 37th FOCS, 1996.
3. Y. Bartal: *On approximating arbitrary metrics by tree metrics*, Proceedings of the 30th STOC, 1998.
4. M. Charikar, C. Chekuri, A. Goel and S. Guha: *Rounding Via Trees: Deterministic Approximation Algorithms for Group Steiner Trees and k-Median*, Proceedings of the 30th STOC, 1998.
5. Goran Konjevod, R. Ravi, Aravind Srinivasan: *Approximation Algorithms for the Covering Steiner Problem*, Proceedings of the 11th SODA, 2000.
6. Jiri Matousek: *Lectures on Discrete Geometry*, Springer, 2002.
7. N. Linial, E. London and Yu. Rabinovich: *The geometry of graphs and some of its algorithmic applications*, Combinatorica, 15, 1995, pp. 215 - 245.
8. Naveen Garg, Goran Konjevod, R. Ravi: *A polylogarithmic approximation algorithm for the group Steiner tree problem*, Journal of Algorithms, 37(1), 2000, pp. 66–84.
9. Uriel Feige: *Approximating the bandwidth via volume respecting embeddings*, Journal of Computer and System Sciences, 60(3), 2000, pp. 510–539.
10. Y.Rabinovich, R.Raz: *Lower Bounds on the Distortion of Embedding Finite Metric Spaces in Graphs*, Discrete and Computational Geometry, 19, 1998, pp. 79-94.

Nearest Neighbors Search Using Point Location in Balls with Applications to Approximate Voronoi Decompositions

Yogish Sabharwal, Nishant Sharma, and Sandeep Sen

Department of Computer Science and Engineering
I.I.T. Delhi-110016, India
{yogish,nishantt,ssen}@cse.iitd.ernet.in

Abstract. We present improved reductions of the nearest neighbor searching problem to Point Location in Balls by constructing linear size Approximate Voronoi Diagrams while maintaining the logarithmic search time. We do this first by simplifying the construction of Har-Peled[9] that reduces the number of balls generated by a logarithmic factor to $O(n \log n)$. We further reduce the number of balls by a new hierarchical decomposition scheme and a generalization of PLEBs to achieve linear space decomposition for nearest neighbor searching.

1 Introduction

Let P be a set of n points in a metric space. One of the most fundamental problems with diverse applications is to build a data structure on P that supports nearest neighbor searching efficiently. This problem has satisfactory solutions when P is planar but becomes non-trivial in higher dimensional Euclidean space even in three dimensions [5,8]. The most general approach to nearest neighbor searching is to build a Voronoi diagram of P (to be denoted as $Vor(P)$) which is a partition of the space into cells such that a cell consists of all points that are closest to a specific point of P than any other point of P. Despite its numerous applications, Voronoi diagrams suffer from the drawback of high structural complexity (and consequent computational bottleneck). It is known that the worst case complexity in $\mathbb{E}^d$ is $\theta(n^{\lceil d/2 \rceil})$ with constants exponential in d. Therefore building and storing these diagrams becomes computationally infeasible with growing dimensions.

As a result, alternate solutions for nearest neighbor searching in higher dimensions have been a hot topic of research in recent years ([3,7,1,10,12,11]). Since the exact problem seems intimately related to the complexity of Voronoi diagrams, the related relaxed problem of *approximate nearest neighbor search* has gained importance and has also shown promise of practical solutions. Given a constant $\varepsilon < 1$, we want to preprocess the data-set P to get a data structure D, such that given a query point q, we can efficiently return a $p \in P$ such that $\forall p' \in P, \text{dist}(p, q) \leq (1 + \varepsilon)\text{dist}(p', q)$. Here, dist could be any arbitrary metric, and then p is an ε nearest neighbor (ε-NN) of q.

M. Agrawal and A. Seth (Eds.): FSTTCS 2002, LNCS 2556, pp. 311–323, 2002.

In the context of this paper, the the work of Indyk and Motwani [10] holds special significance where they reduced the problem of ε-NNS to Approximate Point Location in Equal Balls (ε-PLEB).

Definition 1. *Given P and a parameter r, an ε-PLEB(P, r) is a data structure which when given a query point q returns a point $p \in P$ such that $dist(p, q) \leq r$, if there exists one. If there is no point $p' \in P$ with $dist(p', q) \leq r(1 + \varepsilon)$, then it returns* nil. *Otherwise, it can return anything. When $\varepsilon = 0$, then it is called PLEB(P, r).*

However, their reduction was quite complex which was improved in all respects in the work of Har-Peled [9] who presented an alternate decomposition of space called an *Approximate Voronoi Diagram* (to be referred as AVD) that supports approximate nearest neighbor searching and has a significantly lower space complexity. His result can be summarized as follows.

Theorem 1 (Har-Peled). *Given P of n points in $\mathbb{R}^d$ and a parameter $\varepsilon > 0$ one can compute a set $\mathcal{C}(P)$ of $O(n \frac{\log n}{\varepsilon^d} \log n/\varepsilon)$ regions where each region is a cube or an annulus of cubes. The regions of $\mathcal{C}(P)$ are disjoint and cover the space and each region has an associated point $p \in P$, so that for any point $q \in c$, the point p is a ε-NN of q in P and it can be computed in $O(\log(n/\varepsilon))$ steps.*
The time for constructing the data-structure is $O(n \frac{\log n}{\varepsilon^d} \log^2 n/\varepsilon)$

Har-Peled also used a more sophisticated data structure for solving ε-NNS based on the BBD data-structure of Arya et al[9].

1.1 Our Results

An important question left open in Har-Peled's work was if AVD could be made linear-sized rather than *near linear*. In this paper, we achieve reduction in space by extending some of the ideas of Har-Peled in addition and generalizing the notion of PLEBs to *Point Location in Smallest Ball* (PLSB) where the balls may be of different radii. We also define its approximate version - for precise definitions, the reader may refer to section 3. Har Peled [9] had also addressed this problem implicitly - in a more ad-hoc manner.

Note that PLEB is a special case of PLSB, with all balls of equal radius.

Lemma 1. *An ε-PLSB on n balls can be solved for d-dimensional space endowed with metric l_p, in $O(\log(n))$ query time and $O(n/\varepsilon^d)$ space.*

The method is similar to Har-Peled[9]. We replace each ball of radius r by the set of cubes from the hierarchical grid, of edge length $\frac{r\varepsilon}{d}$, which intersect that ball. Then we construct a compressed quadtree over the cubes generated. To answer a query, among all cubes containing the query point, we find the cube corresponding to the smallest ball. Then we return the ball of the input data set, to which it corresponds. It is easily shown that this solves the ε-PLSB problem for any l_p metric.
In the remaining paper we will only bound the number of balls in the data structure which is linearly related to the space complexity.

In the next section, we eliminate the recursive structure of the Har-Peled's algorithm by constructing a set of balls directly from the clustering and also reduce the number of balls by a factor of $\log n$. The search time remains logarithmic and the preprocessing time is also improved by a factor of $O(\log n)$.

In section 3, we improve the space bound to linear using a new hierarchical clustering scheme that is also based on the MST of P. To reduce the preprocessing time (the exact MST construction takes nearly quadratic time), we use a faster approximate MST algorithm that takes $O(n \log n)$ time - the same as Har-Peled. In Har-Peled's data structure we always merge two clusters, if the MST edge connecting them is smallest. We merge the clusters such that diameter of the new cluster generated is small. We illustrate the difference with an example in section 3

In an independent work, Arya and Malmatos[2] have also achieved a similar result based on the well separated space decomposition of Callahan and Kosaraju [6]. Further, they obtain space-time trade-offs for generalization of the Voronoi cells with more than one representative - this has been further improved by Arya at al. [4]. One possible advantage of our method is that the number of balls generated in our algorithm is much less - $O(n)$ as compared to $O((\alpha\sqrt{d})^d n)$ and an alternate search structure (other than the BBD tree) may be able to exploit this.

2 Improving the Space Bound

Har-Peled [9], proposes a solution based on the following observation:

Observation 1 *: If the distance between two points A and B is $|AB|$, and a query point Q lies outside d-balls of radius $k * AB$, where k is set to $1/\epsilon$, then A and B are ϵ- neighbors of Q, i.e.,* $\frac{|QA|}{|QB|} < 1 + \epsilon$

Har-Peled [9] further proposes a method for clustering the points into a hierarchical tree. Consider the connected components obtained by growing balls around all points. We also construct a forest (called cluster tree) from the connected component.

Initially we start with n connected components consisting of the individual points. The corresponding forest consists of n trees, each tree consisting of a single leaf, corresponding to a point. We then grow the balls around all the points uniformly. When two components meet, they are replaced by a single merged component. In the corresponding forest, the two corresponding trees are merged by hanging one of the trees on the other one arbitrarily (see Figure 1). At this point, we also associate a value of $r_{loss}(p)$ with the root, p of the tree that we hang on the other one. This value corresponds to the radius of the balls when these components meet.
Thus $r_{loss}(p)$ = radius of the ball at p, when p ceases to be root.

Properties of the cluster tree

– Values of r_{loss} increase along a path towards the root.
– If L_{max} is the longest edge of the MST, the tree edge corresponding to this is the last edge to be added.

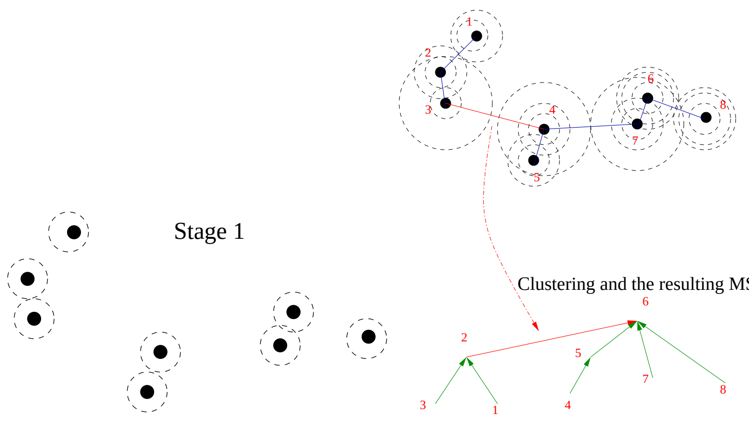


Fig. 1. Constructing the cluster tree

– If any query point q that is outside of the union of balls of radii $L_{max}.|P|.1/\epsilon$ centered at $p \in P$, then any point p is an ϵ-nearest neighbor of q, i.e., q is too far from P. Note that the diameter of P is bounded by $L_{max}.|P|$.

These properties also hold for subtrees (clusters) formed during the construction. Har-Peled, instead of working with an exact MST in d-dimensions which takes $O(n^2)$ time to compute for large d, settles for an nd factor approximation that can be computed in $O(n\log n)$ time.
Using the nd-MST, he constructs an nd-stretch Hierarchical Clustering and then gives an algorithm for the construction of a family of PLEBs (Point Location in Equal Balls) for answering the approximate nearest neighbor queries in time $O(\log(\frac{n}{\epsilon}))$ using space $O(\frac{n}{\epsilon^d}\log n\log(\frac{n}{\epsilon}))$. The bound on the space depends on the number of balls generated by the family of PLEBs.
In this section we present a different perspective, whereby we decrease the number of balls required for answering approximate nearest neighbor queries by pruning overlapping and unwanted balls from the set of balls centered around a point $p \in P$. We give an independent algorithm for the construction of balls around the points of P, such that reporting the center of the smallest ball that contains a query point answers the approximate nearest neighbor query.

2.1 Building the nd-Hierarchical Clustering

The following definitions are similar to Har-Peled that have been restated for convenience of the reader.

Definition 2. *$\lambda - MST$: A tree T having the points of P in its nodes, is a $\lambda - MST$ of P, if it is a spanning tree of P, having a value $len(.)$ associated with each edge of T, such that for any edge $e = uv$ of T, $len(e) \leq d(P_1, P_2) \leq \lambda len(e)$, where P_1,$P2$ is the partition of P into two sets as induced by the two connected connected components of T e, and $d(P_1, P_2) = min_{x \in P_1, y \in P_2}$ $||xy||$ is the distance between P_1 and P_2.*

One can compute a nd-MST of P in $O(n\log n)$. Har-Peled [9] shows how such an nd-MST can be constructed, and is similar to the fair-split tree construction of Callahan and Kosaraju [6].

Definition 3. λ-stretch Hierarchical Clustering (λ-SHC) *: A Directed tree T storing the points of P in it's nodes, so that for any point p, there is a value $r_{loss}(p)$ associated with the out-edge emanating from p to it's parent, satisfying the following property : If C_1 denotes connected components obtained on constructing balls of radius r around all points, C_2 denotes connected components obtained on constructing balls of radius λr around all points, F is the forest obtained by removing from T, all edges larger than r and X is the partition of P into subsets induced by the connected components of the forest F, then $C_1 \leq X \leq C_2$, where $X \leq Y \Rightarrow$ if A and B are in same component in X, then they are in same component in Y.*

The nd-SHC is built from the nd-MST as follows:
We sort the edges of the nd-MST in order of the $len(.)$ values of the edges. We then traverse the list. In each iteration, we take the next edge, say $e = xy$, from the sorted list and connect the two subtrees to which x and y belong by hanging the smaller tree onto the larger one by introducing an edge between the roots of the two subtrees. We associate with this edge, the value $r_{loss}(e) = len(xy)/2$. For root, t, of the tree, we assign $r_{loss}(t) = max$ ($r_{loss}(p_i)$ / p_i is a child of t).

Observation 2 *: The height of the tree formed in the λ-SHC of P is $O(\log n)$.*

Definition 4. *r_{death}, r_{low}, r_{high}: For an approximation factor γ, Define $r_{death}(p) = 6\lambda r_{loss}(p)n\log n/\gamma$. Define $r_{low}(p) = (1/(1+\gamma/3))r_{loss}(p)$. This gives us a ball just smaller than $r_{loss}(p)$, s.t., if a query point $q \notin b(p, r_{low}(p))$ but $q \in b(p, r_{loss}(p))$, then p is a $(1+\gamma/3)$-NN of the query point. Define $r_{high}(p) = (36\lambda r_{loss}(p)n\log n/\gamma)$. This values is defined for a point. Note that $r_{high}(p) > r_{death}(p)$.*

Definition 5. *$parent(p)$, $parent^i(p)$: Define $parent(p)$ to be the Parent of node p in the tree obtained by the λ-SHC of P. Define $parent^i(p)$ to be the i^{th} Parent of node p in the tree obtained by the λ-SHC of P, i.e., $Parent^0(p) = p$ and $Parent^i(p) = parent(parent^{(i-1)}(p))$. Note that $Parent^i(p)$ is defined till it is the root of the tree obtained by the λ-SHC of P, and is not defined beyond the root. We will use $b(p, r)$ to denote the ball of radius r centered at p and $Subtree(p)$ to denote the subtree rooted at p.*

2.2 Construction of Balls (Algorithm $\boldsymbol{ConstructBalls(P,\gamma)}$)

In this section $\lambda = nd$. Given a λ-SHC of P, for each point p in P, let r_0 be the r_{loss} value for p and let p_1, p_2, ...,p_m be the children of p in sorted order of their r_{loss} values, i.e., $r_{loss}(p_1) \leq r_{loss}(p_2) \leq ... \leq r_{loss}(p_m)$. Also, let x be the Parent of p in the tree formed by the λ-SHC of P.
We construct the following ball sets around p:

1. Balls with radius $r_i = r_{loss}(p)(1+\gamma/3)^{(j-1)}$ for $j = 0, .., M-1$, where $M = \lceil \log(1+\gamma/3)(r_{high}(p)/r_{loss}(p)) \rceil$. This defines a ball set in the range $[r_{low}(p), r_{high}(p)]$.
2. Balls with radius $r_i = r_{loss}(p_i)(1+\gamma/3)^{(j-1)}$ for $j = 0, .., M-1$, where $M = \lceil \log(1+\gamma/3)(r_{high}(p_i)/r_{loss}(p_i)) \rceil$. This defines a ball set in the range $[r_{low}(p_i), r_{high}(p_i)]$ $\forall$ $1 \leq i \leq m$.

We also construct a universal ball (of infinite radius) centered at the point that is the root of the tree formed by the λ -SHC of P, so that any point that is not in any of the balls constructed by the above rules lies in this ball.

2.3 Correctly Answering $(1+\epsilon)$-Approximate NN Queries

In this subsection we prove that reporting the center of the smallest ball that contains a query point from the the set of balls constructed by the algorithm ConstructBalls answers the approximate nearest neighbor query correctly.
In the following lemma, $r_{high}(x)$ defines a limit on the radius of the largest ball to be constructed around the point x, so that if the query point does not lie in this ball, then $parent(x)$ is a near neighbor of the query point with some accumulated approximation error. Thus, we can ignore the point x at the expense of some accumulated approximation error. By the definition of $r_{high}(x)$, the accumulated error is a factor of $1+\gamma/(3\log n)$. This value is so chosen that repeated accumulation of approximation errors is bounded by a suitable value.

Lemma 2. *For any query point q, if x is a $(1+\alpha)$-NN of q in P, and if $||xq|| > r_{high}(x)$, then $parent(x)$ is a $((1+\gamma/(3\log n))(1+\alpha))$-NN of q in P.*
Proof omitted.

In the following lemma, we show that it is possible to recursively consider the parent of the current candidate as the next nearest neighbor candidate if the query point does not lie in the largest ball around the current candidate. Of course, at every step that we shift the candidate to the parent, we accumulate an extra error in our approximation.

Lemma 3. *Let p be the NN of a query point q in P. Let r be the radius of the smallest ball containing q and let it be centered at x. If $r > r_{high}(parent^i(p))$ $\forall$ $0 \leq i \leq j-1$, then $parent^j(p)$ is a $((1+\gamma/(3\log n))^j$-approximate NN of q in P.*

Proof. The proof is by induction on j using Lemma 2.

In the following lemma, we show that since we are constructing balls in a manner that every ball is larger by a factor of $1+\gamma/3$ from the previously constructed ball, we may incur an additional approximation error factor of $1+\gamma/3$ in reporting the nearest neighbor. (Note that the stated scenario is possible due to our construction of a discrete set of balls).

Lemma 4. *For any query point q, if p is a $(1+\alpha)$-NN of q in P, and if there exists a ballset, centered at p, in $[r_{low}(t), r_{high}(t)]$ for some point t, and if the*

smallest ball containing q has radius r, such that $r_{loss}(t) \leq r \leq r_{high}(t)$ and is centered at x, then x is a $(1 + \gamma/3)(1 + \alpha)$-approximate NN of q in P.
Proof omitted.

Suppose that the query point q is contained in the ball of radius $r_{loss}(p)$ for a point p. Then, clearly the nearest-neighbor of the query point q can only lie amongst the points that belong to the cluster formed by the subtree of the hierarchical clustering tree rooted at p. The following lemma shows that the construction of balls in algorithm ConstructBalls for this case leads to answering the queries with only a further compounded approximation error of a factor of $1 + \gamma/3$.

Lemma 5. *For any query point q, if p is a $(1 + \alpha)$-NN of q in P, and we have balls constructed around the points of P as defined by ConstructBalls, and if the smallest ball containing q has radius r, such that $r < r_{loss}(p)$ and is centered at x, then x is a $(1 + \gamma/3)(1 + \alpha)$-approximate NN of q in P.*
Proof omitted.
Using the above lemmas, we get the following result.

Theorem 2. *For a point set P and an approximation factor ϵ, Let the smallest ball containing q, in the balls formed by ConstructBalls(P, γ), where $\gamma = \epsilon/2$, be of radius r, centered at x, then x is a $(1 + \epsilon)$-approximate NN of q in P.*
Proof omitted.

2.4 A Bound on the Total Number of Balls Required

Theorem 3. *The total number of balls constructed by the algorithm ConstructBalls$(P, \epsilon/2)$ is $O((1/\epsilon)n\log(n/\epsilon))$.*
Proof omitted.

This improves the bound on the number of balls constructed in [9] by a factor of $\log n$. These balls can be used to construct cells, i.e., quadtree boxes, and then store them in a BBD-tree as described in [9] resulting in an improvement of a factor of $\log n$ in space complexity and pre-processing time while keeping the query time logarithmic.

3 Reduction of ε-NNS to ε-PLSB Based on a New Hierarchical Clustering

First we give definition of *Point Location in Smallest Ball* (PLSB) problem and its approximate version *Approximate Point Location in Smallest Ball*(ε-PLSB).

Definition 6. *1.* **PLSB** *Given a set of points P, and a finite set $Q \subset P \times \mathbb{R}$, we want to build a data structure D, such that given any query point q, we are able to return a pair $(p, r) \in Q$, such that $b(p, r)$ is the smallest ball containing q. If there is no such ball, then it should return NO.*

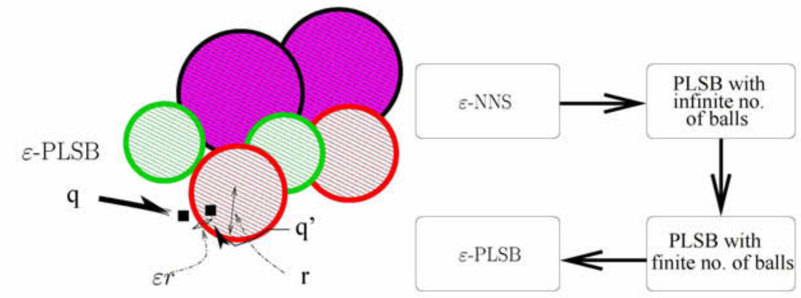

ε-PLSB
q
q'
εr
r
ε-NNS
PLSB with infinite no. of balls
PLSB with finite no. of balls
ε-PLSB

Fig. 2. Strategy

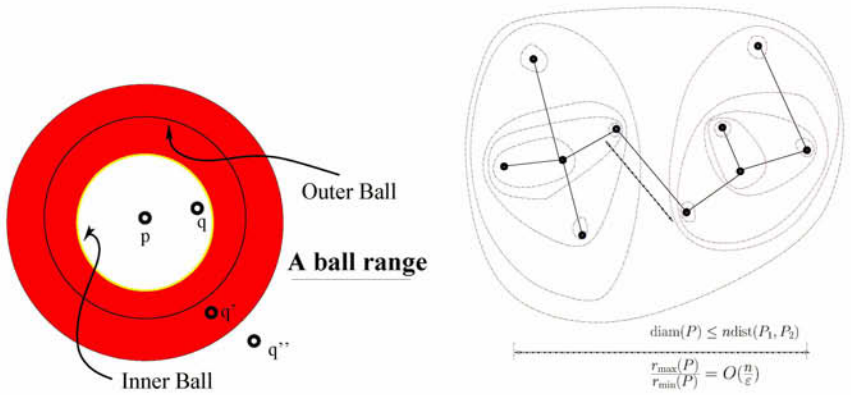

Outer Ball
p
q
A ball range
q'
q''
Inner Ball
$\mathrm{diam}(P) \leq n\,\mathrm{dist}(P_1, P_2)$
$\frac{r_{\max}(P)}{r_{\min}(P)} = O(\frac{n}{\varepsilon})$

Fig. 3. (i)Inner vs. Outer ball (ii) MST hierarchical clustering

2. **ε-PLSB** *Given a set of points P, and a finite set $Q \subset P \times \mathbb{R}$, we want to build a data structure D, such that given any query point q —*
(i) *If $q \in b(p, r)$ with $(p, r) \in Q$. Then we must return a pair $(p', r') \in Q$, such that there exists a point q' such that $dist(q, q') \leq r'\varepsilon$ and $b(p', r')$ is the smallest ball containing q' among balls from $\{b(p, r)|(p, r) \in Q\}$.*
(ii) *If q is not contained in any ball from the set $\{b(p, r(1 + \varepsilon))|(p, r) \in Q\}$, then we should return NIL.*
(iii) *Otherwise, we can return anything(even NIL).*

Given a set of points P, we are trying to construct a set of balls B around points in P, so as to reduce the problem of approximate nearest neighbor searching in P to ε-PLSB in B. For that, we follow strategy described by the figure 2 (b).

The definition of our hierarchical clustering is actually quite generic. It is a binary tree with singleton subsets of P(the input point set) as leafs. The internal nodes are denoted by the union of leafs of the sub-tree rooted at that node. Given ANY such tree, we construct an instance I of ε-PLSB, such that any nearest neighbor query for the point set P can be solved by solving the query q for I.

For two sets X and Y, we denote $\min_{x \in X, y \in Y} \text{dist}(x, y)$ by $\text{dist}(X, Y)$. Using this notation, we define the hierarchical clustering as follows.

Definition 7 (Hierarchical Clustering). *Given a point set P, its hierarchical clustering is a $\underline{\text{binary}}$ tree with singleton subsets of P as leafs(each singleton subset of P appearing once and only once). The internal nodes are denoted by the union of the leafs of the sub-tree rooted at them. Each node in the tree is thus a subset of P and is called a cluster. Note that the root is the cluster P. Each internal node v of the tree is tagged with two values $r_{min}(v)$ and $r_{max}(v)$ with the following properties.*

1. *r_{min} values increase from leafs to root.*
2. *$r_{min}(v) \leq \frac{1}{2} dist(left(v), right(v))$*
3. *$r_{max}(v) \geq 8 \frac{diam(v)}{\varepsilon}$*

Definition 8. *For every cluster v of T, we pick an arbitrary point $p \in v$ and call it leader(v).*

Definition 9 (BallRange(p, r_1, r_2)). *It is defined to be the set of all balls centered at p of radius between r_1 and r_2.* $\text{BallRange}(p, r_1, r_2) = \{b(p, r) | r_1 \leq r \leq r_2\}$ *Note that there are infinite number of balls in* $\text{BallRange}(p, r_1, r_2)$.

Definition 10 (BallSet(v)). *For a cluster v of a hierarchical clustering T, we define BallSet(v) to be*

– *If v is leaf, then BallSet$(v) = \phi$.*
– *Otherwise, BallSet(v) = BallSet$(left(v)) \cup$ BallSet$(right(v)) \cup B_l \cup B_r$ where*
  $B_l = \text{BallRange}(leader(left(v)), r_{min}(v), r_{max}(v))$ *and*
  $B_r = \text{BallRange}(leader(right(v)), r_{min}(v), r_{max}(v))$

Definition 11 (Inner Ball versus Outer ball). *Note that BallSet(v) is actually a union of $2|v| - 2$ BallRanges. A ball is called an inner ball, if it is the smallest ball for one of the BallRanges of BallSet(v). Otherwise, it is called an outer ball.*

Observation 3 *If $b(p, r) \in$ BallSet(v) is an inner ball of BallSet(v), then $r = r_{min}(u)$, for u equal to v or some descendant of v in the hierarchical clustering of v.*

Observation 4 *Suppose, the ball $b(p, r)$ is the smallest ball in BallSet(v) containing q. If $b(p, r)$ is an outer ball, then dist$(q, p) = r$.*

3.1 Strategy

Given a hierarchical clustering, we give the strategy of reduction from ε-NNS to ε-PLSB for an arbitrary metric. This process is hypothetically divided in three steps. In the first step, we generate an infinite set of balls P_1, such that result of PLSB query q to P_1 is a ball, centered around $\varepsilon/7$-nearest neighbor of q (If none exists then all points in P are $\varepsilon/7$- nearest neighbors of q). In the next step, we generate a finite set of balls P_2 from P_1, such that result of PLSB query q to P_2 is a ball centered around $2\varepsilon/3$-nearest neighbor of q. Finally, we prove that the result of $\varepsilon/7$-PLSB query q to P_2 is a ball centered around ε-nearest neighbor of q in P.

Generating P_1(Reduction to PLSB with Infinite No. of Balls). Here we have to reduce $\varepsilon/7$-NNS to PLSB with infinite number of balls. For that, we prove that for any cluster v of T, BallSet(v) is such a point set. The exact statement is the theorem 4.

Theorem 4. *For a cluster v of T, given a query point q, a point $p \in v$ is $\varepsilon/7$ nearest neighbor of q among points of v, if one of the following is true.*

- *If p is the center of the smallest ball in BallSet(v) containing q.*
- *There is no ball in BallSet(v) containing q.*

Note that the second condition depends only on q, and so any arbitrary database point p is $\varepsilon/7$-nearest in that case.

Proof. We prove it by induction on the height of v in T. In the base case, v is a leaf, and BallSet(v) is empty. So, it is trivially proved.

Now, in the induction step, we assume the result for left(v) and right(v), and prove it for v. Let $p_1 = \text{leader}(\text{left}(v))$ and $p_2 = \text{leader}(\text{right}(v))$. By definition, $\text{BallSet}(v) = (\text{BallSet}(\text{left}(v)) \cup B_l) \cup (\text{BallSet}(\text{right}(v)) \cup B_r)$, where B_l and B_r are a set of balls around p_1 and p_2 respectively. Note that $\text{BallSet}(\text{left}(v)) \cup B_l = B_1$(say) is a set of balls with centers among points of left(v) and BallSet(right(v)) $\cup B_r = B_2$(say) is a set of balls with centers among points of right(v). We consider three cases(exhaustive but not necessarily disjoint).

Case 1 *There exists no ball in B_1 containing q* — Since, $b(p_1, r_{\max}(v)) \in B_1$, so $\text{dist}(q, p_1) \geq r_{\max}(v) \geq 8\text{diam}(v)/\varepsilon$. So, distance between q and p_1 is large compared to the diameter of v. It is easily shown that for any point $p \in v$ $\frac{\text{dist}(q,p)}{\text{dist}(q,v)} \leq \frac{r_{\max}(v)}{r_{\max}(v)-\text{diam}(v)} \leq \frac{8}{8-\varepsilon} \leq 1 + \varepsilon/7$

Case 2 *There exists no ball in B_2 containing q* — Similar to case 1.

Case 3 *There exists a ball each in B_1 and B_2 containing q* — Let, the smallest ball in B_1 and B_2 containing q be $b(p_1^*, r_1)$ and $b(p_2^*, r_2)$ respectively. To handle this case, we use the following lemma.

Lemma 6. *p_1^* is an $\varepsilon/7$-nearest neighbor of q, among points of left(v), and p_2^* is an $\varepsilon/7$-nearest neighbor of q, among points of right(v).*

The proof of the lemma 6 is omitted. Now we consider the three sub-cases arising in case 3. We have $\text{dist}(q, p_1^*) \le r_1$ and $\text{dist}(q, p_2^*) \le r_2$.
Case 3a *The ball $b(p_1^*, r_1)$ is an inner ball* —In this case, from observation 3, we have $r_1 = r_{\min}(v')$ for some $v' \subseteq v$. Using this with the definition of $r_{\min}$ we get :- $\text{dist}(q, p_1^*) \le r_1 = r_{\min}(v') \le r_{\min}(v) \le 0.5\text{dist}(\text{left}(v), \text{right}(v))$ This implies that $\text{dist}(q, \text{left}(v)) \le 0.5\text{dist}(\text{left}(v), \text{right}(v))$ But, $\text{dist}(q, \text{right}(v)) \ge \text{dist}(\text{left}(v), \text{right}(v)) - \text{dist}(q, \text{left}(v))$ Therefore, $\text{dist}(q, \text{right}(v)) \ge 0.5\text{dist}(\text{left}(v), \text{right}(v)) \ge \text{dist}(q, \text{left}(v))$ So, $\text{dist}(q, v) = \text{dist}(q, \text{left}(v))$ Now, $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,v)} = \frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{left}(v))} \le 1{+}\varepsilon/7$ So, p_1^* is an $\varepsilon/7$ nearest neighbor of q in v.
Also, any ball in B_2 containing q must have a radius larger than $\text{dist}(q, \text{right}(v))$. But, $\text{dist}(q, \text{right}(v)) \ge 0.5\text{dist}(\text{left}(v), \text{right}(v)) \ge r_{\min}(v')$ $= r_1$. So, any ball in B_2 containing q is larger than $b(p_1^*, r_1)$. So, we see that the smallest ball containing q in BallSet(v) is centered around $\varepsilon/7$ nearest neighbor of q in v. **Case 3b** *The ball $b(p_2^*, r_2)$ is an inner ball* — Similar to case 3a.
Case 3c *Both the balls $b(p_2^*, r_2)$ and $b(p_2^*, r_2)$ are outer balls* — From observation 4, we have $r_1 = \text{dist}(q, p_1^*)$ and $r_2 = \text{dist}(q, p_2^*)$. Without loss of generality, $r_1 \le r_2$. So, $b(p_1^*, r_1)$ is the smallest ball containing q, in $B_1 \cup B_2 = \text{BallSet}(v)$.
Also, $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{left}(v))} \le 1 + \varepsilon/7$ and $\frac{\text{dist}(q,p_2^*)}{\text{dist}(q,\text{right}(v))} \le 1 + \varepsilon/7$ from the lemma 6. Since, p_1^* is nearer to q than p_2^*, therefore $\frac{\text{dist}(q,p_1^*)}{\text{dist}(q,\text{right}(v))} \le 1 + \varepsilon/7$. Therefore $\frac{\text{dist}(q,p_1^*)}{min(\text{dist}(q,\text{left}(v)),\text{dist}(q,\text{right}(v)))} \le 1{+}\varepsilon/7$ This implies that p_1^* is the $\varepsilon/7$ nearest neighbor of q in v.

Generating P_2 from P_1(Reduction to $\varepsilon/7$-PLSB with Finite No. of Balls). In this section, we show that we can drop most of the balls of P_1, retaining only a finite number of balls. For that we consider each ball range separately.

Theorem 5. *Given a hierarchical clustering $T = (V, E)$ for a point set P, we can generate a set of $2\sum_{v \in V} \max(1, \log_{1+\varepsilon} \frac{r_{max}(v)}{r_{min}(v)})$ balls such that we can construct an ε-PLSB for them with approximation factor $1+\varepsilon/7$, such that, given a query point q, the point returned by PLSB data structure is $1{+}\varepsilon$ nearest neighbor of q among points of P.*

Proof. We start with BallSet(P). Since, there are infinite number of balls, we need to drop some of them. Recall that balls in the BallSet(P) can be expressed as union of $2|P| - 2$ BallRanges. We replace each BallRange$(p, r_1, r_2) \subset$ BallSet(P) with $\left\{ b(p,r) | r = r_1 \left(1 + \frac{\varepsilon}{7}\right)^k \wedge b(p,r) \in \text{BallSet}(P) \right\}$.

Call this smaller set of balls, B. Then, It is easy to show that, a ε-PLSB constructed for B with approximate factor $\varepsilon/7$ returns a ball which is centered around $(1 + \varepsilon/7)(1 + \varepsilon/7)(1 + \varepsilon/7) \le (1 + \varepsilon))$ nearest neighbor.

B is composed of ball ranges, two for each cluster. Number of balls in ball-ranges corresponding to cluster v is $2\max\left(1, \log_{1+\varepsilon}\frac{r_{\max}(v)}{r_{\min}(v)}\right)$. So, we need at most $2\sum_{v\in V}\max\left(1, \log_{1+\varepsilon}\frac{r_{\max}(v)}{r_{\min}(v)}\right)$ balls.

3.2 Constructing the Hierarchical Clustering

Given a point set P, we want to construct a hierarchical clustering such that $2\sum_{v\in V}\max\left(1, \log_{1+\varepsilon}\frac{r_{\max}(v)}{r_{\min}(v)}\right) = O\left(|P|\log_{1+\varepsilon}(1/\varepsilon)\right)$. The first hierarchical clustering that we describe is directly based on the MST of the point set. Suppose, the MST of the point set P is $M = (P, E'_M)$. Then we construct the hierarchical clustering as follows — The longest edge in E'_M divides P into two parts, say P_1 and P_2 respectively. We find the hierarchical clustering of P_1 and P_2 recursively, say C_1 and C_2. Then, the MST hierarchical clustering of P has P as root and C_1 and C_2 as two children of the root. Suppose, v is a cluster in the MST hierarchical clustering. We use $r_{\min}(v) = \frac{1}{2}\text{dist}(\text{left}(v), \text{right}(v))$, and $r_{\max}(v) = 8\text{diam}(v)/\varepsilon$. Note that, this definition satisfies the properties 1,2 and 3 of the definition 7

It is easily seen that for MST hierarchical clustering,
$2\sum_{v\in V}\max\left(1, \log\frac{r_{\max}(v)}{r_{\min}(v)}\right) = O\left(|P|\log_{1+\varepsilon}(|P|/\varepsilon)\right)$ The problems with MST hierarchical clustering are the following:-

1. The preprocessing involves construction of the MST, which requires nearly quadratic time.
2. The preprocessing involves finding diameters of point sets which is computationally expensive.
3. The space requirement is not linear but $n\log n$.

To handle the first problem, we can use approximate MSTs instead of exact MST. To handle the second problem, we can use alternative definition of $r_{\max}(v)$. To handle third problem, we shall have to construct the hierarchical clustering from the MST(or approximate MST) in a different manner. In this version, we omit the details for constructing the hierarchical clustering from approxmate MST. We summarize the following result (without proof).

Theorem 6. *Given P of n points in $\mathbb{R}^d$ and a parameter $\varepsilon > 0$ one can compute a set $\mathcal{C}(P)$ of $O(\frac{n}{\varepsilon^d}\log_{1+\varepsilon}\frac{1}{\varepsilon})$ regions where each region is a cube or an annulus of cubes. The regions of $\mathcal{C}(P)$ are disjoint and cover the space and each region has an associated point $p \in P$, so that for any point $q \in c$, the point p is an ε-NN of q in P and it can be computed in $O(d\log\frac{n}{\varepsilon})$ steps.*
The construction can be done in $O(npolylog(n))$.

References

1. S. Arya, D.M. Mount, N.S. Netanyahu, R. Silverman, and A.Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

2. Sunil Arya and Theocharis Malamatos. Linear-size approximate voronoi diagrams. *to appear in Proceedings of SODA*, 2002.
3. Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimension. *Proc. of SIAM-ACM SODA*, pages 271 – 280, 1993.
4. Sunil Arya Theocharis Malamatos and David Mount. Space-efficient approximate voronoi diagrams. *Proceedings of ACM STOC*, 2002.
5. D. Attali and J.D. Boissonnat. Complexity of the delau- nay triangulation of points on a smooth surface. *http://www-sop.inria.fr/prisme/personnel/boissonnat/papers.html*, 2001.
6. P.B. Callahan and S.R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential elds. *J. ACM*, pages 42:67–90, 1995.
7. Kenneth L. Clarkson. An algorithm for approximate closest-point queries. *In Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 160–164, 1994.
8. J. Erickson. Nice points sets can have nasty delaunay triangulations. *In Proc. 17th Annu. ACM Sympos. Comput. Geom..*, 2001.
9. S. Har-Peled. A replacement for voronoi diagrams of near linear size. *Proc of IEEE FOCS pages 94 – 103, 2001.*
10. P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. *In Proc. 30th Annu. ACM Sympos. Theory Comput.*, pages 604–613, 1998.
11. Piotr Indyk. *PhD Thesis.* PhD thesis, Stanford University, 1999.
12. Eyal Kushilevitz, Rafail Ostrovsky, and Yuval Rabani. Efficient search for approximate nearest neighbour search in high dimensional spaces. In *ACM Symposium on Theory of Computing*, pages 614–623, 1998.

Formal Languages and Algorithms for Similarity Based Retrieval from Sequence Databases

A. Prasad Sistla

Dept. of Computer Science, Univ. of Illinois at Chicago
sistla@cs.uic.edu

Abstract. Similarity based retrieval is of major importance for querying sequence databases. We consider formalisms based on automata, temporal logics and regular expressions for querying such databases. We define two different types of similarity measures—syntax based and semantics based. These measures are divided into a spectrum of measures based on the vector distance function that is employed. We consider norm vector distance functions and give efficient query processing algorithms when these measures are employed.

1 Introduction

Recently there has been much interest in similarity based retrieval from sequence databases. The works in this area, consider a database of sequences and provide methods for retrieving sequences that approximately match a given query. Such methods can be applied for retrieval from time-series, video and textual databases. Earlier work in this area [FRM94] considered the case when the query is given by a single sequence and developed fast methods for retrieving all database sequence that closely match the query sequence. In this paper, we consider the case when the query is given by a predicate over sequences specified in various formalisms, and give efficient methods for checking if a database sequence approximately satisfies the query predicate.

We consider formalisms based on automata, temporal logic, and regular expressions for specifying queries over sequences. We define similarity based semantics for these formalisms. More specifically, for a database sequence d and query q, we define a similarity measure $sim(d, q)$ that denotes how closely d satisfies the query q. This measure ranges from zero to one denoting the different levels of satisfaction; higher values denote greater levels of satisfaction with value one indicating perfect satisfaction. Actually, we define $sim(d, q)$ to be $1 - dist(d, q)$ where $dist(d, q)$ is a distance measure between d and q.

For example, in a database consisting of daily stock sequences, one might request a query such as— "retrieve all the daily stock patterns in which IBM stock price remained below 70 until the Dow-Jones value reached 10,000". Such a query can be expressed in Temporal Logic (or any of the other formalisms) by considering "IBM stock price is less than 70" and "Dow-Jones value equals 10,000" as atomic propositions. An answer to such a query will not only return sequences that exactly satisfy the temporal predicate, but also those sequences that satisfy it approximately, i.e., those sequences having a similarity value greater than a given threshold specified by the user.

The distance measures that we define are classified in to semantics based and syntax based measures. We define two types of semantic based distance measures. In both of these types, the distance measure is defined using the exact semantics of the query, which is given by a set S of sequences. When the query q is given by an automaton then S is the set of sequences accepted by the automaton; when q is a temporal logic formula, then S is the set of (finite) sequences that satisfy the formula; when q is a regular expression, then S is the language specified by the regular expression. The first semantics based distance measure between d and q, is defined to be the minimum of the vector distances between d and each sequence in the set S. The second distance semantic measure is more complex and is based on replacing certain symbols in sequences of S by the wild card symbols (section 2 contains the actual definition). By using various norm vector distance functions, we get a spectrum of the two types of semantic distance measures.

The syntax based distance measure is defined only for the cases when the query is specified by a temporal formula or by a regular expression. In this case,the distance measure is defined inductively based on the syntax of the query, i.e. the $dist(d, q)$ is defined as a function of the distance measures of d with respect to the top level components of q (i.e., sub-formulas when q is a temporal logic formula). For example, the syntax distance with respect to the temporal formulas $g \wedge h$ is defined to be the maximum of the distances with respect to g and h. We relate the syntax and semantics based distance measures.

We present algorithms for computing the syntactic and semantic distances for a given database sequence and a query. For the case when the query is given by an automaton or by a regular expression, the algorithms for the first semantic distance measure have linear complexity in the size of the automaton and polynomial complexity in the length of the sequence (actually, the complexity is linear in the length of the sequence for the infinite norm, and is quadratic for other cases); for the second semantic distance measure the algorithms have the same complexity with respect to the length of the sequence, but have triple exponential complexity in terms of the automaton size. When the query is given by a temporal logic formula, the algorithms for the semantic distance measures have the same complexity with the following exception: the first semantic distance measure has exponential complexity in the length of the formula; this blow up is caused by the translation of the temporal formula in to an automaton.

The algorithms for computing syntactic distance measures have linear complexity in the length of the query and polynomial complexity in the length of the database sequence; (more specifically, the complexity with respect to the database sequence is linear for the infinite norm vector distance function, and is quadratic in other cases).

The paper is organized as follows. Section 2 gives definitions and notation. Section 3 reviews and presents algorithms for the case when the query is specified by automata. Section 4 defines the distance measures for temporal logic and presents algorithms for computing these. Section 5 presents the corresponding results for the regular expressions. Section 6 briefly discusses related work. Section 7 contains conclusions.

2 Definition and Notation

In this section, we define the various formalisms that we consider in the paper and their similarity based semantics. For a sequence $s = (s_0, s_1, ..., s_i, ...s_{n-1})$, we let $\mathtt{s[i]}$ denote its suffix starting from s_i. We let $|s|$ denote the length of s. If s, t are sequences, then we let st denote the concatenation of s and t in that order. We represent a sequence having only one element by that element. A sequence over a set Δ is a sequence whose elements are from Δ. A language L over Δ is a set of sequences over Δ. We let Δ^* represent the set of all such sequences.

Let Σ be a set of elements, called atomic queries. Each member of Σ represents an atomic query on a database state. With each database state u and atomic query a, we associate a similarity value $simval(u, a)$ that denotes how closely d satisfies a. This value can be any value between zero and one (one indicates perfect satisfaction). We use a special atomic query $\phi \notin \Sigma$, called wild card, which is always satisfied in every database state; that is, $simval(u, \phi) = 1$ for every database state u. Let $d = (d_0, d_1, ..., d_{n-1})$ be a sequence of database states and $a = (a_0, ..., a_{n-1})$ be a sequence over $\Sigma \cup \{\phi\}$. Corresponding to d and a, we define a sequence of real numbers called $simvec(d, a)$ defined as follows. Let $i_0 < i_1 < ... < i_{m-1}$ be all values of j such that $0 \leq j < n$ and $a_j \neq \phi$. We define $simvec(d, a)$ to be the sequence $(x_0, ..., x_{m-1})$ where $x_j = simval(d_{i_j}, a_{i_j})$ for $0 \leq j < m$. Intuitively, we define $simvec(d, a)$ by ignoring the positions corresponding to the wild card symbol. It is to be noted that if a contains only ϕ then $simvec(d, a)$ is the empty sequence, i.e. it is of length zero.

Let F be a distance measure over real vectors assigning a positive real value less than or equal to one, i.e., F is a function which associates a real value $F(\boldsymbol{x}, \boldsymbol{y})$, such that $0 \leq F(\boldsymbol{x}, \boldsymbol{y}) \leq 1$, with every pair of real vectors $\boldsymbol{x}, \boldsymbol{y}$. We assume that $F(\boldsymbol{x}, \boldsymbol{y}) = 1$ whenever $\boldsymbol{x}, \boldsymbol{y}$ are not of the same length and $F(\boldsymbol{x}, \boldsymbol{y}) = 0$ when $\boldsymbol{x}, \boldsymbol{y}$ are the empty vectors. Given a database sequence d and a sequence a over $\Sigma \cup \{\phi\}$, we define $dist(d, a, F)$ to be $F(simvec(d, a), \mathbf{1})$ where $\mathbf{1}$ denotes a vector all of whose components are 1.

Let L be a language over Σ, d be a database sequence, and F be a vector distance function. We define two distance measures, $distance_1(d, L, F)$ and $distance_2(d, L, F)$, of d with respect to L using the vector distance function F. The value of $distance_1(d, L, F)$ is defined to be the minimum of $\{dist(d, a, F) : a \in L\}$.

The definition of $distance_2(d, L, F)$ is more complex and is motivated by the following situation. Suppose each atomic query in Σ denotes a logical predicate and the disjunction of all these predicates is a tautology, i.e. it is always satisfied. As an example, consider the case when $\Sigma = \{P, \neg P\}$ where P is an atomic proposition. Now consider the language $L_1 = \{aP : a \in \Sigma\}$. The language L_1 requires that the atomic proposition P be satisfied at the second state irrespective of whether P is satisfied or not at the first state. It can be argued that the distance of a database sequence d with respect to L_1 should depend only on the distance of the second database state with respect to P. This intuition leads us to the following definitions.

Let $closure(L)$ be the smallest language L' over $\Sigma \cup \{\phi\}$ such that $L \subseteq L'$ and the following closure condition is satisfied for every $\alpha, \beta \in (\Sigma \cup \{\phi\})^*$: if $\alpha a \beta \in L'$ for every $a \in \Sigma$ then $\alpha \phi \beta \in L'$. (Recall ϕ is the wild card symbol). Now we define a partial order $<$ on the set of sequences over $\Sigma \cup \{\phi\}$ as follows. Let $\alpha = (\alpha_0, ..., \alpha_{m-1})$ and $\beta = (\beta_0, ..., \beta_{n-1})$ be any two sequences over $\Sigma \cup \{\phi\}$. Intuitively, $\alpha < \beta$ if α can be obtained from β by replacing some of the occurrences of the wild card symbol ϕ in β by a symbol in Σ. Formally, $\alpha < \beta$ iff $m = n$ and for each $i = 0, 1, ..., n-1$ either $\alpha_i = \beta_i$ or $\beta_i = \phi$, and there exists at least one value of j such that $\alpha_j \in \Sigma$ and $\beta_j = \phi$.

It is not difficult to see that $<$ is a partial order. Let S be any set of sequences over $\Sigma \cup \{\phi\}$. A sequence $\alpha \in S$ is called maximal if there does not exist any other sequence $\beta \in S$ such that $\alpha < \beta$. Let $maximal(S)$ denote the set of all maximal sequences in S. Now we define $distance_2(d, L, F)$ to be the value of $distance_1(d,\ maximal(closure(L)), F)$. For the language L_1 (given above) $maximal(closure(L))$ consists of the single sequence ϕP; for a database sequence d of length two, it should be easy to see that $distance_2(d, L_1, F)$ equals the distance of the second database state with respect to P.

It is to be noted that the distance functions $distance_1$ and $distance_2$ depend on the vector distance function F. We consider a spectrum of vector distance functions $\{F_m : m = 1, 2, ..., \infty\}$ defined as follows. Let $\boldsymbol{x}, \boldsymbol{y}$ be two vectors. For each $m = 1, ..., \infty$, if $\boldsymbol{x}, \boldsymbol{y}$ are of unequal length then $F_m(\boldsymbol{x}, \boldsymbol{y}) = 1$. Otherwise, let n be the length of both $\boldsymbol{x}$ and $\boldsymbol{y}$. In this case, $F_m(\boldsymbol{x}, \boldsymbol{y})$ is given as follows. For $m \neq \infty$,

$$F_m(\boldsymbol{x}, \boldsymbol{y}) = \left(\frac{\Sigma_{1 \le i \le n}(|x_i - y_i|)^m}{n}\right)^{1/m} \tag{1}$$

$$F_\infty(\boldsymbol{x}, \boldsymbol{y}) = \max\{|x_i - y_i| \ : \ 1 \le i \le n\} \tag{2}$$

Note that F_1 is the average block distance function and F_2 is the mean square distance function, etc. It can easily be shown that $F_\infty(\boldsymbol{x}, \boldsymbol{y}) = \lim_{m \to \infty} F_m(\boldsymbol{x}, \boldsymbol{y})$. Note that $F_1(\boldsymbol{x}, \boldsymbol{y})$ gives equal importance to all components of the vectors; however, as m increases, the numerator in the expression for $F_m(\boldsymbol{x}, \boldsymbol{y})$ is dominated by the term having the maximum value, i.e. by $\max\{|x_i - y_i| \ : \ 1 \le i \le n\}$, and in the limit $F_\infty(\boldsymbol{x}, \boldsymbol{y})$ equals this maximum value. Thus we see that F_1 and F_∞ are the extremes of our distance functions. We call F_1 as the *average block distance* function and F_∞ as the *infinite norm* distance function.

The following lemma can easily be proved from our earlier observations.

Lemma 1. *For every database sequence d and language L over Σ the following properties hold.*

1. $distance_2(d, L, F_\infty) \le distance_1(d, L, F_\infty)$.
2. *For any $i, j \in \{1, 2, ..., \infty\}$ such that $i < j$, $distance_1(d, L, F_i) \le distance_1(d, L, F_j)$ and $distance_2(d, L, F_i) \le distance_2(d, L, F_j)$.*

3 Automata

In this section, we consider automata for specifying queries over sequences. We give algorithm for computing the distance of a database sequence with respect to a given automaton.

An automaton $\mathcal{A}$ is 5-tuple $(Q, \Sigma, \delta, I, F)$ where Q is a finite set of states, Σ is a finite set of symbols called the input alphabet, δ is the set of transitions, $I, F \subseteq Q$ are the set of initial and final states, respectively. Each transition of $\mathcal{A}$, i.e. each member of δ, is a triple of the form (q, a, q') where $q, q' \in Q$ and $a \in \Sigma$; this triple denotes that the automaton makes a transition from state q to q' on input a; we also represent such a transition as $q \ \rightarrow_a \ q'$. Each input symbol represents an atomic predicate (also called an atomic query in some places) on a single database state. For example, in a stock market database, $price(ibm) = 100$ represents an atomic predicate. In a textual database, each database state represents a document and a database sequence represents a sequence of documents; here an atomic predicate may state that the document contain some given key words. In a video database, which is a sequence of images or shots, each atomic predicate represents a condition on a picture such as requiring that the picture contain some given objects.

Let $a = a_0, a_1, ..., a_{n-1}$ be a sequence of input symbols from Σ and q, q' be states in Q. We say that the sequence a takes the automaton $\mathcal{A}$ from state q to q' if there exists a sequence of states $q_0, q_1, ..., q_n$ such that $q_0 = q$ and $q_n = q'$ and for each $i = 0, ..., n-1$, $q_i \rightarrow_{a_i} q_{i+1}$ is a transition of $\mathcal{A}$. For any state q, we let $T(q)$ denote the set of sequences that take the automaton from state q to a final state. We say that the automaton $\mathcal{A}$ accepts the string a if there exists an initial state q such that $a \in T(q)$. We let $L(\mathcal{A})$ denote the set of strings accepted by $\mathcal{A}$.

We identify vectors with sequences. We let $\mathbf{1}$ denote a vector all of whose components are 1. The length of the vector will be clear from the context.

For a database sequence d, automaton $\mathcal{A}$, and a vector distance function F, we define two distances $distance_i(d, \mathcal{A}, F)$, for $i = 1, 2$ as follows: $distance_i(d, \mathcal{A}, F) = distance_i(d, L(\mathcal{A}), F)$.

Algorithms for Computing the distances

Now we outline an algorithm for computing the value of $distance_1(d, \mathcal{A}, F_\infty)$. Recall that F_∞ is the infinity norm vector distance measure. Let $d = (d_1, ..., d_n)$ be the database sequence. Essentially, the algorithm computes the distances of the suffixes of d with respect to the states of the automaton for increasing lengths of the suffixes. For any automaton state q and integer i $(1 \leq i \leq n)$, the algorithm first computes the value of $distance_1(\texttt{d[i]}, T(q), F_\infty)$ in decreasing values of i starting with $i = n$. (Recall that `d[i]` is the suffix of d starting with d_i and $T(q)$ is the set of sequences accepted by $\mathcal{A}$ starting in the state q). The algorithm finally computes $distance_1(d, \mathcal{A}, F_\infty)$ to be minimum of the values in $\{distance(\texttt{d[1]}, T(q), F_\infty) \ : q \in I\}$; note that `d[1]` is simply d. The values in the set $\{distance_1(\texttt{d[i]}, T(q), F_\infty) \ : q \in Q\}$ are computed in decreasing values of i using the recurrence equation given by the following lemma which is proved by induction using the definition of the distance function $distance_1$.

Lemma 2. *Let q be any state in Q and i be an integer such that $1 \leq i \leq n$. Further, let $q \rightarrow_{a_1} q_1, ..., q \rightarrow_{a_m} q_m$ be all the transitions in δ from the state q. Then the following properties hold.*

1. *For $i < n$, $distance_1(\texttt{d[i]}, T(q), F_\infty) = \min\{x_1, ..., x_m\}$ where $x_j = \max\{(1 - simval(d_i, a_j)), distance_1(\texttt{d[i+1]}, T(q_j), F_\infty)\}$ for $j = 1, ..., m$.*
2. *If there is at least one j such that $q_j \in F$ then $distance_1(q, \texttt{d[n]}, F_\infty) = \min\{(1 - simval(d_n, a_j)) : q_j \in F\}$, otherwise $distance_1(q, \texttt{d[n]}, F_\infty) = 1$.*

It is easy to see that the complexity of this algorithm is $O(n|\mathcal{A}|)$ where $|\mathcal{A}|$ is the size of the automaton $\mathcal{A}$. Thus the algorithm is of linear complexity in n and $|\mathcal{A}|$. A formal description of the algorithm is given in [HS00].

The above algorithm can be modified to compute $distance_1(d, \mathcal{A}, F_i)$ for any i such that $0 < i < \infty$. The algorithm is based on some of the techniques given in [HS00]. This algorithm is of complexity $O(n^2|\mathcal{A}|)$.

Computing the value of $distance_2(d, \mathcal{A}, F)$

Now, we show how to compute the values of $distance_2(d, \mathcal{A}, F)$. From the definition, we have $distance_2(d, \mathcal{A}, F) = distance_1(d, maximal(closure(L(\mathcal{A}))), F)$. Let $\Sigma' = \Sigma \cup \{\phi\}$ where ϕ is the wild card symbol. Recall that the elements of $closure(L(\mathcal{A}))$ are strings over the alphabet Σ'. We construct an automaton $\mathcal{H}$ that accepts the language $maximal(closure(L\mathcal{A}))$. Then we simply compute $distance_2(d, \mathcal{A}, F)$ to be the value of $distance_1(d, \mathcal{H}, F)$. While computing the later value using the equations of 2, we take $simval(d_i, \phi)$ to be 1.

Now we show how to compute the automaton $\mathcal{H}$. First we compute the automaton $\overline{\mathcal{A}}$ which accepts the complement of the language accepted by the automaton $\mathcal{A}$. From this automaton, we construct another automaton $\mathcal{C}$ which accepts the complement of the language $closure(L(\mathcal{A}))$. $\mathcal{C}$ simulates $\overline{\mathcal{A}}$ on an input string with the following modification. Whenever it sees the input symbol ϕ, it replaces it, non-deterministically, by some symbol from Σ and simulates $\overline{\mathcal{A}}$ on the guessed symbol. It accepts it if $\overline{\mathcal{A}}$ accepts. It is not difficult to see that $\mathcal{C}$ accepts a string σ over the alphabet Σ' if there exists a string σ', obtained by replacing the ϕ symbols in σ by some symbol from Σ, which is accepted by $\overline{\mathcal{A}}$. Hence, it is easy to see that $\mathcal{C}$ accepts the complement of the language $closure(L(\mathcal{A}))$. Next we construct the automaton $\overline{\mathcal{C}}$ which accepts the complement of the language accepted by $\mathcal{C}$, i.e., which accepts the language $closure(L(\mathcal{A}))$. It is to be noted that in the worst case the sizes of $\overline{\mathcal{C}}$ and $\overline{\mathcal{A}}$ can be exponential in the sizes of $\mathcal{C}$ and $\mathcal{A}$, respectively. As a consequence the size of $\overline{\mathcal{C}}$ can be double exponential in the size of $\mathcal{A}$.

Using $\overline{\mathcal{C}}$, we construct an automaton $\mathcal{D}$ which accepts the complement of the language $maximal(closure(L(\mathcal{A}))$. The automaton $\mathcal{D}$ on an input string σ over Σ' acts as follows. It accepts σ if either $\sigma \notin closure(L(\mathcal{A}))$, or there exists another string $\sigma' \in closure(L(\mathcal{A}))$ such that $\sigma < \sigma'$ (here $<$ is the partial order on strings defined in section 2). The former condition is checked by simulating $\mathcal{C}$ over σ. To check the later condition, $\mathcal{D}$ non-deterministically changes at least one of the input symbols in σ, which is an element of Σ, to ϕ and checks that the resulting string is accepted by $\overline{\mathcal{C}}$, i.e., is in $closure(L(\mathcal{A}))$. It is easy to see that we can construct such an automaton $\mathcal{D}$ such that its size is linear in the sizes of $\mathcal{C}$ and $\overline{\mathcal{C}}$, and hence is double exponential in the size of $\mathcal{A}$. Next we construct the complement $\overline{\mathcal{D}}$ of $\mathcal{D}$. Clearly $\overline{\mathcal{D}}$ accepts the language $maximal(closure(\mathcal{A}))$. We take $\mathcal{H}$ to be the automaton $\overline{\mathcal{D}}$. Clearly, its size is triple exponential in the size of $\mathcal{A}$.

For each $i = 1, ..., \infty$, we compute $distance_2(d, \mathcal{A}, F_i)$ to be $distance_1(d, \mathcal{H}, F_i)$. For $i = \infty$, the complexity of the algorithm is linear in the length of d and triple exponential in the size of $\mathcal{A}$. For $i < \infty$, the complexity is quadratic in the length of d and triple exponential in the size of $\mathcal{A}$.

4 Temporal Logic

In this section, we consider linear Temporal logics as one of the formalism for specifying queries over database sequences. Such logics have been extensively used in specification of properties concurrent programs [MP92]. They have also been used in database systems for specifying queries in Temporal databases [Ch092a] and for specifying triggers in active database systems [SW95a]. We assume that we have a finite set $\mathcal{P}$ whose members are called atomic propositions. Each member of this set denotes an atomic predicate over a database state. Formulas of Temporal Logics (TL) are formed from atomic propositions using the propositional connectives $\wedge, \vee, \neg$ and the temporal operators $\mathsf{Nexttime}$ ("nexttime") and Until ("until"). The set of formulas of TL is the smallest set satisfying the following conditions. Every atomic proposition is a formula of TL; if g and h are formulas of TL then $g \wedge h$, $g \vee h$, $\neg g$, $\mathsf{Nexttime}\ g$ and $g\ \mathsf{Until}\ h$ are also formulas of TL.

Given a database sequence $d = (d_0, d_1, ..., d_{n-1})$ and a temporal formula f, and a vector distance function F, we define a distance function $syndist(d, f, F)$ inductively based on the syntax of f as follows.

- For an atomic proposition P, $syndist(d, P, F) = 1 - simval(d_0, P)$.
- $syndist(d, g \wedge h, F) = \max\{syndist(d, g, F), syndist(d, h, F)\}$.
- $syndist(d, g \vee h, F) = \min\{syndist(d, g, F), syndist(d, h, F)\}$.
- $syndist(d, \neg g) = 1 - syndist(d, g, F)$.
- $syndist(d, \mathsf{Nexttime}\ g, F) = syndist(\mathtt{d[1]}, g, F)$.
- $syndist(d, g\ \mathsf{Until}\ h, F) = \min\{F(\boldsymbol{U_i}, \mathbf{1}) : 0 \leq i < n\}$ where $\boldsymbol{U_i}$ is the vector $(u_{i,0}, u_{i,1}, .., u_{i,i})$ whose components are given as follows. $u_{i,i} = 1 - syndist(\mathtt{d[i]}, h, F)$ and for j, $0 \leq j < i$, $u_{i,j} = 1 - syndist(\mathtt{d[j]}, g, F)$. Intuitively, this definition corresponds to the exact semantics of Until .

Now, we define two types of semantic distance functions between a database sequence d and a TL formula f. Let $\Delta = 2^{\mathcal{P}}$ and $s = (s_0, ..., s_{n-1})$ be any sequence over Δ. We define the satisfaction of f at the beginning of s inductively on the structure of f as follows. For an atomic proposition $P \in \mathcal{P}$, s satisfies P if $P \in s_0$. s satisfies $\mathsf{Nexttime}\ g$ if $n > 0$ and $\mathtt{s[1]}$ satisfies g. s satisfies $g\ \mathsf{Until}\ h$ if there exists an $i < n$ such that $\mathtt{s[i]}$ satisfies h and for all j, $0 \leq j < i$, $\mathtt{s[j]}$

satisfies g. The satisfaction, for the cases when f is a conjunction, disjunction or negation of formulas, is defined in the standard way.

We say that two TL formulas f and g are equivalent if the sets of sequences that satisfy them are identical. Let $\mathcal{P} = \{P_0, P_1, ..., P_{m-1}\}$ and $s = (s_0, ..., s_{n-1})$ be any sequence over Δ. Let Σ be the set consisting of the elements of $\mathcal{P}$ and negations of elements in $\mathcal{P}$. Formally, $\Sigma = \mathcal{P} \cup \{\neg P_i : 0 \leq i < m\}$. Now we define a sequence $expn(s)$ over Σ which is obtained from s by expanding each s_i in to a subsequence of length m whose j^{th} element is P_j or $\neg P_j$ depending on whether P_j is in s_i or not. Formally, $expn(s) = (t_0, ..., t_{nm-1})$ is a sequence of length nm defined as follows: for each i, j such that $0 \leq i < n$ and $0 \leq j < m$, if $P_j \in s_i$ then $t_{im+j} = P_j$, otherwise $t_{im+j} = \neg P_j$.

For a TL formula f, let $L(f) = \{expn(s) : s \text{ satisfies } f\}$. Note that $L(f)$ is a language over Σ.

Let $d = (d_0, d_1, ..., d_{n-1})$ be a database sequence. Now, we define another database sequence e obtained from d by repeating each d_i successively m times (recall m is the number of atomic propositions), i.e., $e = ((d_0)^m, (d_1)^m, ..., (d_i)^m, ..., (d_{n-1})^m))$. Let f be a TL formula and F be a vector distance function.. Now, for each $j = 1, 2$, we define a semantic distance of d with respect to f and F (denoted by $semdist_j(d, f, F)$) as follows: $semdist_j(d, f, F) = distance_j(d, L(f), F)$. Recall that $distance_j$ is defined in the previous subsection.

It is to be noted that the syntactic distances (i.e., $syndist(d, f, F)$) as well as all the semantic distances (i.e., $semdist_j(d, f, F)$) only depend on the similarity values of atomic propositions that appear in f and not on other atomic propositions. For the syntactic distance, this fact follows directly from the definition. However, for the semantic distance this fact can be proven from the way we defined this distance. The semantic distances of a database sequence with respect to equivalent TL formulas are equal. However this property does not hold for syntactic distances. For example, the syntactic distance of a database sequence with respect to the two equivalent formulas $(P \wedge Q) \vee (P \wedge \neg Q)$ and P may be different. The following lemma shows that $syndist(d, f, F_\infty) \leq semdist_2(d, f, F_\infty)$. The lemma can be proven by induction on the structure of the formula f.

Lemma 3. *For any database sequence d and TL formula f in which all negations are applied to atomic propositions, $syndist(d, f, F_\infty) \leq semdist_2(d, f, F_\infty)$.*

Algorithm for computing the distances

Let $d = (d_0, ..., d_{n-1})$ be a database sequence and f be a TL formula. Now we present algorithms for computing the syntactic and the semantic distances of d with respect to f. First we consider the syntactic distance. We assume that we are given the similarity values of the database states in d with respect to the atomic propositions appearing in f. Let $SF(f)$ be the set of all subformulas of f. For each $g \in SF(f)$ and for each $i = 0, ..., n-1$, we compute $syndist(\mathtt{d[i]}, g, F)$ inductively on the structure of g as follows. The method is straightforward, from the definition, for the cases when g is an atomic proposition, or is of the form $g_1 \wedge g_2$, $g_1 \vee g_2$, $\neg g_1$, $\mathsf{Nexttime}\ g_1$. For the case when

$g = g_1$ Until g_2 we do as follows.. We compute $\mathit{syndist}(\mathtt{d[i]}, g, F)$ for decreasing values of i. We first give the method for the case when $F = F_\infty$. For $i = n-1$, this value is computed to be $\mathit{syndist}(\mathtt{d[i]}, g_2, F_\infty)$. For $i < n-1$, $\mathit{syndist}(\mathtt{d[i]}, g, F_\infty)$ is computed to be the minimum of the two values—$\mathit{syndist}(\mathtt{d[i]}, g_2, F_\infty)$ and $\max\{\mathit{syndist}(\mathtt{d[i]}, g_1, F_\infty),\ \mathit{syndist}(\mathtt{d[i+1]}, g, F_\infty)\}$. It is easy to see that this procedure only takes $O(n)$ time. For $k \neq \infty$, we compute the values $\{\mathit{syndist}(\mathtt{d[i]}, g, F_k) : 0 \le i < n\}$ from the values $\{\mathit{syndist}((\mathtt{d[i]}, g_1, F_k), \mathit{syndist}(\mathtt{d[i]}, g_2, F_k) : 0 \le i < n\}$ in a straightforward way from the definitions; it is easy to see that this algorithm can be implemented in in $O(n^2)$ time.

The complexity of the algorithm for computing $\mathit{syndist}(d, f, F_\infty)$ is $O(|f||d|)$ where $|f|$ is the length of the formula and $|d|$ is the length of d. The complexity of the algorithms for $\mathit{syndist}(d, f, F_k)$ (for $k \neq \infty$) is $O(|d|^2|f|)$.

Now we consider the computation of the semantic distances of d with respect to f. We take an automata theoretic approach for computing $\mathit{semdist}_k(d, f, F_i)$ for $i = 1, ..., \infty$. For this we use the well known result (see [VWS83,ES83]) that shows that there exists an automaton $\mathcal{A}$ whose size is $O(2^{c|f|})$ such that $L(\mathcal{A}) = L(f)$. The value of $\mathit{symdist}_1(d, f, F_i)$ is computed as the value $\mathit{distance}_1(d, \mathcal{A}, F_i)$ using the algorithm given in [HS00].

To compute the value of $\mathit{semdist}_2(d, f, F_i)$, we use the approach given in section 3. This approach uses the complement $\overline{\mathcal{A}}$ of the automaton $\mathcal{A}$. The automaton $\overline{\mathcal{A}}$ accepts the language $\overline{L(\mathcal{A})}$, i.e., the complement of the language $L(\mathcal{A})$; this is exactly the set of sequences that satisfy the formula $\neg f$. Thus we can take $\overline{\mathcal{A}}$ to be the automaton that accepts the set of all sequences that satisfy $\neg f$. Using the approach given in [VWS83,ES83] we can obtain such an automaton whose size is $O(2^{c|f|})$ for some small constant $c > 0$. Using this automaton, we can use the procedure given in section 3. The complexity of the resulting algorithm will be $O(|d|2^{2^{2^{c|f|}}})$ for the case when we use the distance function F_∞; for all other distance functions F_i ($i < \infty$), the complexity is $O(|d|^2 2^{2^{2^{c|f|}}})$.

5 Regular Expressions

In this section we consider regular expressions (REs) as query languages and define syntactic and semantic distances of a database sequence with respect to REs. Let Σ be a finite set of atomic queries. The set of REs over Σ is the smallest set of strings satisfying the following conditions. Every element of Σ is a RE; if g and h are REs then $(g \vee h), gh$ and $(g)^*$ are also REs. With each RE f over Σ, we associate a language $L(f)$ over Σ defined inductively as follows. For every $a \in \Sigma$, $L(a) = \{a\}$. For the REs g, h, $L(gh) = L(g)L(h)$, $L(g \vee h) = L(g) \cup L(h)$, $L((g)^*) = (L(g))^*$.

Let $d = (d_0, ..., d_{n-1})$ be a database sequence and f be a RE. For each $k = 1, ..., \infty$ we define a syntactic distance function, denoted by $\mathit{syndist}(d, f, F_k)$ inductively on the structure of f as follows. First we define this for the case when $k \neq \infty$.

- For $a \in \Sigma$, $\mathit{syndist}(d, a, F_k) = 1 - \mathit{simval}(d_0, a)$ when $n = 1$, i.e. $|d| = 1$; otherwise $\mathit{syndist}(d, a, F_k) = 1$.

- $syndist(d, g \vee h, F_k) = \min\{syndist(d, g, F_k),\ syndist(d, h, F_k)\}$.
- $syndist(d, gh, F_k) = \min\{(\frac{|\alpha| u^k + |\beta| v^k}{n})^{\frac{1}{k}} : \alpha \in L(g),\ \beta \in L(h)$ are non-empty database sequences such that $\alpha\beta = d$ and $u = syndist(\alpha, g, F_k)$ and $v = syndist(\beta, h, F_k)\}$.
- We define $syndist(d, (g)^*, F_k)$ as follows. For any database sequence β, let $Val(\beta) = |\beta|(syndist(\beta, g, F_k))^k$. We define $syndist(d, (g)^*, F_k)$ to be $\min\{(\frac{Val(\alpha_1)+\ldots+Val(\alpha_l)}{n})^{\frac{1}{k}} : \alpha_1\alpha_2\ldots\alpha_l = d$ and for $1 \leq i \leq l$, $\alpha_i \in L(g)$ and is non-empty$\}$

The values of $syndist(d, f, F_\infty)$ are defined as follows.

- For $a \in \Sigma$, $syndist(d, a, F_\infty) = 1 - simval(d_0, a)$ when $n = 1$, i.e. $|d| = 1$; otherwise $syndist(d, a, F_\infty) = 1$.
- For the RE $g \vee h$, $syndist(d, g \vee h, F_\infty) = \min\{syndist(d, g, F_\infty), syndist(d, h, F_\infty)\}$.
- $syndist(d, gh, F_\infty) = \min\{\max\{syndist(\alpha, g, F_\infty),\ syndist(\beta, g, F_\infty)\} : \alpha \in L(g),\ \beta \in L(h)$ are non-empty database sequences such that $\alpha\beta = d\ \}$.
- We define $syndist(d, (g)^*, F_\infty)$ as follows. We define $syndist(d, (g)^*, F_\infty)$ to be $\min\{\max\{syndist(\alpha_1, g, F_\infty), \ldots, syndist(\alpha_l, g, F_\infty)\} : \alpha_1\alpha_2\ldots\alpha_l = d$ and each α_i is non-empty$\}$

For each $k = 1, \ldots, \infty$, we define two semantic distance functions $semdist_1$ and $semdist_2$ as follows, For any database sequences d and RE f and for $j = 1, 2$, $semdist_j(d, f, F_k) = distance_j(d, L(f), F_k)$. The following lemma can be easily proven. It shows that the syntactic distance and the semantic distance given by $semdist_1$ are identical.

Lemma 4. *For each databases sequence d and RE f and for each $k = 1, \ldots, \infty$, $syndist(d, f) = semdist_1(d, f, F_k)$.*

For a RE f, let $A(f)$ be a standard non-deterministic automaton that accepts $L(f)$ and such that the size of $A(f)$ is linear in $|f|$ (see [LP98]). The values of $semdist_1(d, f, F_k)$ for each $k = 1, \ldots, \infty$ can be computed by constructing the automaton $A(f)$ (possibly non-deterministic) and using the algorithm given in [HS00]. These algorithms are of complexity $O(|d||f|)$.

Given a database sequence d and RE f, $semdist_2(d, f, F_k)$ is computed exactly on the same lines as given in section 3. First we obtain the automaton $\overline{A}$ that accepts all strings in $\Sigma^* - L(f)$. The size of the resulting automaton will be $O(2^{|f|})$. The reminder of the steps is same as given in the section 3. As before, the complexity of the algorithm is triple exponential in $|f|$ but linear in $|d|$. Because of this complexity, it might be better to use the syntactic distance measure for similarity based retrieval. Note that this distance function is also same as the first semantic distance function $semdist_1$.

6 Related Work

There has been much work done on querying from time-series and other sequence databases. For example, methods for similarity based retrieval from such databases have been proposed in [FRM94,AFS93]. These methods assume that the query is also a single sequence, not a predicate on sequences as we consider here.

There has also been much work done on data-mining over time series data [GRS99] and other databases. These works mostly consider discovery of patterns that have a given minimum level of support. They do not consider similarity based retrieval.

A temporal query language and efficient algorithms for similarity based retrieval have been presented in [SYV97]. That work uses a a syntactic distance measure which is ad hoc. On the contrary, in this work, we consider syntactic as well as semantic distance measures. Further, in this paper, we consider a spectrum of these measures based on well accepted standard norm distance measures on vectors.

There has been work done on approximate pattern matching (see [WM92] for references) based on regular expressions. They use different distance measures. For example, they usually use the edit distance as a measure and look for patterns defined by a given regular expression with in a given edit distance. On the other hand, we consider average measures; for example, the distance function F_1 defines average block distance. In the area of bio-informatics much work has been done on sequence matching (see [D98] for references). Most of this work is based on probabilistic models (such Markov or extended Markov models). They do not employ techniques based on indices for the subsequence search.

Predicates on sequences have been employed in specifying triggers in Active Database Management Systems [D88,GJS92,SW95a]. However, there exact semantics is used for firing and processing the triggers.

Lot of work on fuzzy logic considers assignment of similarity values to propositional formulae based on their syntax. However, to the best of our knowledge no other work has been done for logics on sequences.

7 Conclusions

In this paper, we have considered languages based on automata, temporal logic and regular expressions for specifying queries over sequence databases. We have defined a variety of distance measures, based on the syntax and semantics of the queries. We have outlined algorithms for computing these values. The algorithms for computing syntactic distance measures are only of polynomial time complexity in the length of the query and polynomial in the length of the database sequence. The algorithms for computing the first semantic distance measure have lower complexity than the second semantic distance measure. Thus, from the complexity point of view, it might be better to use the syntactic based measures or the first semantic distance measure. The algorithms for automata have been extended for real databases that support fast retrieval using indices and have been implemented and tested on real data (see [SHC02] for details).

It is to be noted that all the distance measures that we defined are based on norm vector distance functions. We feel these vector distance functions are the most appropriate for the applications mentioned earlier in the paper. On the other hand, other distance functions between sequences, may be appropriate for other applications. For example, the edit distance may be appropriate in applications involving bio-informatics. As part of future work this needs further investigation.

References

[AFS93] Agarwal R., Faloutsis C., Swami A: *Efficient Similarity Search in Sequence Databases*, In FODO Conference, Evanston, Illinois, Oct. 1993.

[Ch092a] J. Chomicki, *History-less Checking of Dynamic Integrity Constraints*, IEEE International Conference on Data Engineering, Phoenix, Arizona, February 1992.

[D88] Dayal U., *Active Database Management Systems*, Proc. of 3rd Intnl. Conf. on Data and Knowledge bases — Improving usability and Responsiveness, Jerusalem, June 1988.

[D98] Durbin R., et al.: *Biological Sequence Analysis*, Cambridge University Press, 1998.

[ES83] E. A. Emerson, A. P. Sistla: Triple Exponential Decision Procedure for the Logic CTL*, Workshop on Logics of Programs, Carnegie-Mellon University, Pittsburgh, Pennsylvania, June 1983.

[FRM94] Faloutsos F., Ranganathan M., Manolopoulos: *Fast Subsequence Matching in Time-Series Databases*, Proc. of the 1994 ACM SIGMOD Intnl. Conf. on Management of Data, Minneapolis, MN, May 1994.

[GJS92] Gehani N., Jagadish H., Shmueli O.: *Composite Event Specification in Active Databases: Models and Implementation*, Proc. 18th Intnl. Conference on Very Large Databases, Aug. 1992.

[GRS99] Garofalakis, M. N., Rastogi R., Shim K.: *SPIRIT: Sequential Pattern Mining with Regular Expression Constraints*, Proc, of the 25th Intnl Conf. on Very Large Databases, Edinburgh, Scotlanad,UK, 1999.

[HS00] Tao Hu, A. Prasad Sistla: *Similarity based Retrieval from Sequence Databases using Automata as Specifications*, Technical report, Dept of Electrical Engg and Computer Sciejce, 2000.

[LP98] H. Lewis and C. Papadimitriou: *Elements of the Theory of Computation*, Prentice-Hall, 1998.

[MP92] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems—Specification*, Springer-Verlag 1992.

[SHC02] A. P. Sistla, T. Hu, V. Chowdhry *Similarity based Retrieval from Sequence Databases Using Automata as Queries* 11th ACM Intnl Conference on Information and Knowledge Management, Virginia, Nov 2002.

[SW95a] A. P. Sistla and O. Wolfson, *Temporal Triggers in Active Databases*, IEEE Transactions on Knowledge and Data Engineering, Vol 7, No 3, June 1995, pp 471-486.

[SYV97] Sistla A. P., Yu C., Venkatasubramanian R.: *Similarity based Retrieval of Videos*, 13th International Conference on Data Engineering, April, 1997, Birmingham, U.K.

[VWS83] M. Vardi, P. Wolper, A. P. Sistla, *Reasoning about Infinite Computations*, IEEE FOCS 1983.

[WM92] Wu S., Manber U.,: *Fast Text Searching: Allowing Errors*, CACM Oct. 1992, pp 83-91.

Decomposition in Asynchronous Circuit Design[⋆]

Walter Vogler[1] and Ralf Wollowski[2]

[1] Institut für Informatik, Universität Augsburg
vogler@informatik.uni-augsburg.de
[2] FB Elektro- und Informationstechnik, Universität Kaiserlautern
wollo@rhrk.uni-kl.de

Abstract. Signal Transition Graphs (STGs) are a version of Petri nets for the specification of asynchronous circuit behaviour. It has been suggested to decompose such a specification as a first step; this leads to a modular implementation, which can support circuit synthesis by possibly avoiding state explosion or allowing the use of library elements. We present a decomposition algorithm and formally prove it correct, where an interesting aspect is the use of a bisimulation with angelic nondeterminism. In contrast to similar approaches in the literature, our algorithm is very generally applicable.

1 Introduction

Signal Transition Graphs (STGs) are a version of Petri nets for the specification of asynchronous circuits, supported by the tools petrify (e.g. [6]) and CASCADE [2]. The transitions are labelled with input or output signals[1]; the latter are thought to be controlled by the circuit, the former by its environment; nevertheless, the occurrence of an input signal in some state might not be specified, formulating the assumption on the environment not to produce this signal.

Being Petri nets, STGs allow a causality-based specification style, and they give a compact representation of the desired behaviour since they represent concurrency explicitly. As a first step in the synthesis of a circuit from a given STG N, one usually constructs the reachability graph, where one might encounter the state explosion problem. To avoid it, one could try to decompose the STG into components C_i; their reachability graphs taken together can be much smaller than the one of N. Even if this is not achieved, several smaller components might be easier to handle: depending on the circuit synthesis method, further steps might easily take quadratic time in the number of states, or the reachability graph of N might even be too large for the available memory space. Decomposition can also be useful independently of size considerations: there are examples where N cannot be handled by a specific synthesis method, while the

⋆ Partially supported by DFG-project 'STG-Dekomposition' Vo615/7-1 / Wo814/1-1.

[1] Usually, the labels in STGs are not signals, but rising and falling edges of signals, which for each signal are required to alternate; this is of minor importance here, so we abstract from this distinction.

C_i can; also, one may be able to split off a library element, and this is valuable in particular for arbiters, which are complicated to synthesize.

[5,9] suggest decomposition methods for STGs, but these approaches can only deal with very restricted net classes. [5] only decomposes live and safe free choice nets, which cannot model controlled choices or arbitration, and makes further restrictions; e.g. each transition label is allowed only once (which makes the STG deterministic), and conflicts can only occur between input signals. [5] constructs for each output signal s a component C_i that generates this signal; C_i has as inputs all signals that – according to the net structure – may directly cause s. The component is obtained from the STG N by contracting all transitions belonging to the signals that are neither input nor output signals for this component. This contraction is required to be tr-preserving (as defined in [5]), and it might be necessary to add further signals to the inputs to ensure this.

In [5], it is stated that the parallel composition of the C_i – i.e. the (modular) *implementation* – has the same language as N; in the restricted setting of [5], this is the same as having isomorphic reachability graphs. Clearly, this is very strict and not a necessary requirement for an implementation to be correct. On the other hand, language equivalence is too weak in general, since it ignores which choices are possible during a run, and in particular it ignores deadlocks; it seems that in general some form of bisimilarity would be more suitable. The formal proofs for the correctness statement in [5] are very involved.

A similar decomposition method is described in [9]; only marked graphs (which have no conflicts at all) with only output signals are considered and the treatment is quite informal. In contrast to [5], a component can generate several output signals and different components can generate the same signal; this gives more flexibility, but additional components are needed to collect occurrences of the same signal generated by different components. Further, rather informal considerations of this decomposition method can be found in [3].

In this paper, we have a fresh look at the decomposition problem. In particular, we will suggest a method where there are no restrictions on the graph-theoretic structure of the given STG N; to some degree we will even deal with arc weights greater 1 and unsafe nets, which can be useful as we will demonstrate in an example. There are restrictions on the labelling, e.g. conflicts between input and output signals are not allowed, since they prevent the synthesis of a reliable digital circuit. STGs are required to be deterministic, but very importantly, we allow several transitions to have the same label.

Our method is based on [5], but components may generate several output signals. As [5], we suggest to apply mainly *secure* t-contractions, already studied in [1]. Secureness is a part of being tr-preserving as in [5], but in contrast to tr-preservation in general, it is easy to check from the *local* net structure. Thus, only in our version the steps of the decomposition algorithm become efficient.

After presenting basic definitions of STGs in Section 2, we have a closer look at contractions in Section 3, also considering bisimilarity and non-secure contractions. In Section 4, we describe our method in detail. We give a flexible description which allows not only secure contractions to be used but *any* operation that is admissible in some sense; in this sense, our correctness proof is partially reusable. It is thus easier to see that also the deletion of redundant places is admissible, and also non-secure contractions under certain conditions, e.g. if each transition label occurs only once as in [5]. This of course depends on our correctness criterion, which has the following important features.

- We ensure that the composition of the C_i is free of what is called computation interference e.g. in [8], where one component produces an output that is an unspecified input for another; it seems that this problem is ignored in [5].
- We only consider environments behaving as specified by the original STG N; the composition of the components might specify additional inputs, but we ignore these and any subsequent behaviour since they cannot occur if the implementation runs in an appropriate environment. Both these features are not new, see e.g. [7,8], but new in the context of STG decomposition.
- We achieve both these features with a bisimulation-like correctness definition. This style is technically useful in our correctness proof, and it will become even more important for future extensions to nondeterministic STGs. Interestingly, our proof technique is based on a kind of angelic bisimulation, where internal transition occurrences only serve to find a matching behaviour, but are not required to be matched on their own.

The main contribution of this paper is that – transferring the first and second of these features to the area of STG decomposition – we obtain a more generally applicable decomposition method with a much easier correctness proof compared to [5]. We present some examples for our method in Section 5; further examples as well as a supporting tool are in preparation. Further research topics are discussed in the conclusion in Section 6. See [10] for omitted proofs, further explanations including small examples and further references. The authors thank Ben Kangsah for helping with the figures and working out part of the examples.

2 Basic Notions of Signal Transition Graphs

A *Signal Transition Graph* or *STG* is a net that models the desired behaviour of an asynchronous circuit. Its transitions are labelled with signals from some alphabet Σ or with the empty word λ, and we distinguish between input and output signals. A transition labelled with λ represents an internal, unobservable signal, and in this paper, we use such transitions only in intermediate phases of our algorithm.

An *STG* $N = (P, T, W, l, M_N, In, Out)$ is a labelled net consisting of finite disjoint sets P of *places* and T of *transitions*, the *arc weight* $W : P \times T \cup T \times P \to \mathbb{N}_0$, the *labelling* $l : T \to In \cup Out \cup \{\lambda\}$, the *initial marking* $M_N : P \to \mathbb{N}_0$ and the disjoint sets $In \subseteq \Sigma$ and $Out \subseteq \Sigma$ of *input* and *output signals*. If $l(t) \in In$ ($l(t) \in Out$, $l(t) = \lambda$ resp.) then t is an input (an output, an *internal* resp.) transition, drawn as a black (a white resp.) box (as a line). When we introduce an STG N or N_1 etc., then implicitly this introduces its components P, T, ... or P_1, T_1, ... etc.

We assume familiarity with the notions: *arc*, *pre-* and *postset* $^\bullet x$ and $x^\bullet$, *loop*, *tokens* and *marking*; when a transition t or a transition sequence w is *enabled* under a marking M (notation $M[t\rangle$, $M[w\rangle$) and yields M' ($M[t\rangle M'$, $M[w\rangle M'$) when *firing*; *reachable* marking, *safeness* and *boundedness*. Often, STGs are assumed to be safe and to have only arcs with weight 1. In the first place, we are interested in such STGs; but we also deal with others, in particular since they can turn up in our decomposition algorithm.

We can extend the labelling to transition sequences as usual, i.e. $l(t_1 \dots t_n) = l(t_1) \dots l(t_n)$; note that internal signals are automatically deleted. A sequence v of signals from Σ is *enabled* under a marking M, denoted by $M[v\rangle\rangle$, if there is some transition sequence w with $M[w\rangle$ and $l(w) = v$; $M[v\rangle\rangle M'$ is defined analogously. The *language* $L(N)$ is $\{v \mid M_N[v\rangle\rangle\}$, and we call two STGs *language equivalent* if they have the same language.

The idea of input and output signals is that only the latter are under the control of the circuit modelled by an STG. The STG requires that certain outputs are produced provided certain inputs have occurred, namely those outputs that are enabled under the marking reached by the signal occurrences so far. At the same time, the STG describes assumptions about the environment that controls the input signals: if some input signal is not enabled, the environment is supposed not to produce this input at this stage; if it does, the specified system may show arbitrary behaviour, and it might even malfunction.

In this paper, a specification is a *deterministic* STG, i.e. it does not have internal transitions and for each reachable marking and each signal s, there is at most one s-labelled transition enabled under the marking. We distinguish two forms how determinism can be violated.

- If there are two different transitions t_1 and t_2 with the same label $s \in \Sigma$ and a reachable marking M with $M[t_1\rangle$ and $M[t_2\rangle$, then they are *enabled concurrently* and the STG has *auto-concurrency*, if $W(., t_1) + W(., t_2) \leq M$; otherwise they are in (and the STG has) a *dynamic auto-conflict*.
- Two different transitions t_1 and t_2 – and also the signals labelling them – are *in structural conflict* if $^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$. If both transitions are labelled with the same signal $s \in \Sigma$, then the STG has a *structural auto-conflict*. If t_1 is an input (or a λ-labelled) and t_2 an output transition, then they form a *structural input/output conflict* (or a *structural* λ*/output conflict*) and the STG *has* such a conflict.

Clearly, an STG without internal transitions is deterministic if and only if it is without auto-concurrency and without dynamic auto-conflict; the latter is ensured if there are no structural auto-conflicts.

Often, having the same behaviour is understood as language equivalence, but just as often one has to consider the more detailed behaviour equivalence bisimilarity. A *bisimulation* between N_1 and N_2 is a relation $\mathcal{B}$ between markings of N_1 and N_2 such that $(M_{N_1}, M_{N_2}) \in \mathcal{B}$ and for all $(M_1, M_2) \in \mathcal{B}$ we have: if $M_1[t\rangle M_1'$, then there is some M_2' with $M_2[l_1(t)\rangle\rangle M_2'$ and $(M_1', M_2') \in \mathcal{B}$ – and vice versa. If such a bisimulation exists, we call the STGs *bisimilar*.

In a *parallel composition*, the composed systems run in parallel synchronizing on common signals. Since a system controls its outputs, we cannot allow a signal to be an output of more than one component. An output signal of one component can be an input of one or several others, and in any case it is an output of the composition. We will consider computation interference later.

The parallel composition of STGs N_1 and N_2 is defined if $Out_1 \cap Out_2 = \emptyset$. Then, let $A = (In_1 \cup Out_1) \cap (In_2 \cup Out_2)$ be the set of common signals. To get the *parallel composition* $N = N_1 \parallel N_2$, take the disjoint union of N_1 and N_2 and then combine each s-labelled transition of N_1 with each s-labelled transition of N_2 if $s \in A$. Finally, put $In = (In_1 \cup In_2) - (Out_1 \cup Out_2)$ and $Out = Out_1 \cup Out_2$.

Clearly, we can consider markings of the composition as the disjoint union of markings of the components; this makes clear what we mean by the restriction $M\big|_{P_i}$ of a marking M of the composition. It should be clear that, up to isomorphism, composition is associative and commutative. Therefore, we can define the parallel composition of a family (or collection) $(C_i)_{i \in I}$ of STGs as $\|_{i \in I}\, C_i$, provided that no signal is an output signal of more than one of the C_i.

3 Transition Contraction

Transition contraction was e.g. studied in [1]. The first requirement about the arc weights of t is presumably not so essential for our results, compare [1], hence it is more a convenience.

Definition 1. Let N be an STG and $t \in T$ with $W(.,t), W(t,.) \in \{0,1\}$, ${}^\bullet t \cap t^\bullet = \emptyset$ and $l(t) = \lambda$. We define the *t-contraction* $\overline{N}$ of N by

$$\overline{P} = \{(p,*) \mid p \in P - ({}^\bullet t \cup t^\bullet)\} \cup \{(p,p') \mid p \in {}^\bullet t, p' \in t^\bullet\}$$

$$\overline{T} = T - \{t\} \qquad \bar{l} = l\big|_{\overline{T}} \qquad \overline{In} = In \qquad \overline{Out} = Out$$

$$\overline{W}((p,p'),t_1) = W(p,t_1) + W(p',t_1) \quad \overline{W}(t_1,(p,p')) = W(t_1,p) + W(t_1,p')$$

$$M_{\overline{N}}((p,p')) = M_N(p) + M_N(p')$$

Here, $* \notin P \cup T$ is a dummy element with $W(*,t_1) = W(t_1,*) = M_N(*) = 0$.

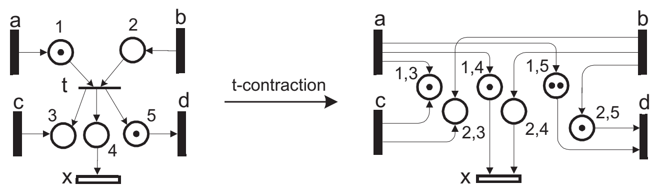


Fig. 1.

Figure 1 shows a part of a net and the result when the internal transition is contracted. In many cases, the preset or the postset of the contracted transition has only one element, and then the result of the contraction looks much easier. We get the following result for t satisfying the preconditions of Definition 1:

Theorem 2. *1.* $L(N) \subseteq L(\overline{N})$ *2. If* $({}^\bullet t)^\bullet \subseteq \{t\}$*, then* N *and* $\overline{N}$ *are bisimilar. 3. If* ${}^\bullet(t^\bullet) = \{t\}$ *and* $\exists p_0 \in t^\bullet : \ M_N(p_0) = 0$*,* N *and* $\overline{N}$ *are language equivalent.*

If the preconditions of 2. or 3. are satisfied, we call the contraction of t secure*; a secure contraction preserves boundedness and freedom from auto-concurrency.*

4 Decomposing a Signal Transition Graph

Given a fixed STG N as a specification of some desired behaviour, we want to decompose it into a collection of components $(C_i)_{i\in I}$ that together implement the specified behaviour, avoiding the complete state exploration for N.

We assume that N is *deterministic*, since in the area of circuit design behaviour is usually specified without any choices. To see the absence of dynamic auto-conflicts easily, we also require N to be *free of structural auto-conflicts*.

We further assume that N is *free of structural input/output conflicts* to ensure that there are no dynamic input/output conflicts, which are very hard to implement: the input, which is under the control of the environment, might occur at roughly the same time as the output, which is under the control of the system, and can therefore not prevent the output as specified. In applications, N will be bounded or even safe; but our results also hold in the general case.

4.1 Correctness Definition

Components $(C_i)_{i\in I}$ are correct when their composition 'somehow' matches the behaviour prescribed by N. As Chu [5], one could require N and $\|_{i\in I} C_i$ to be language equivalent, but this is actually too restrictive; a more liberal notion that is still language based can be found e.g. in [7]. Our idea of correctness is close to the notion of [7,8], but we will define correctness in a bisimulation style.

Definition 3. A collection of deterministic components $(C_i)_{i\in I}$ is a *correct decomposition* or a *correct implementation* of a deterministic STG N, if the parallel composition C of the C_i is defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is a relation $\mathcal{B}$ between the markings of N and those of C with the following:

1. $(M_N, M_C) \in \mathcal{B}$
2. For all $(M, M') \in \mathcal{B}$, we have:

a) If $a \in In_N$ and $M[a\rangle\rangle M_1$, then either $a \in In_C$ and $M'[a\rangle\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M'_1 or $a \notin In_C$ and $(M_1, M') \in \mathcal{B}$.
b) If $x \in Out_N$ and $M[x\rangle\rangle M_1$, then $M'[x\rangle\rangle M'_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M'_1.
c) If $x \in Out_C$ and $M'[x\rangle\rangle M'_1$, then $M[x\rangle\rangle M_1$ and $(M_1, M'_1) \in \mathcal{B}$ for some M_1.
d) If $x \in Out_i$ for some $i \in I$ and $M'\big|_{P_i}[x\rangle\rangle$, then $M'[x\rangle\rangle$. (no *computation interference* [8])

Here and for any collection $(C_i)_{i\in I}$, P_i stands for P_{C_i}, Out_i for Out_{C_i} etc.

In this definition, we allow C to have fewer input and output signals than N, since there might be some outputs that actually never have to be produced (this presumably indicates an error in the specification) or there might be some input signals that are not relevant for any outputs; it is very useful that our algorithm might detect such so-called *globally irrelevant* inputs.

The first three clauses of Part 2 are as usual; the first says that an input allowed by N is also allowed by C (or ignored), the second that specified outputs can be made by C, and the third that it does not make others. Remarkably, there is no clause requiring a match for inputs of C. If $M'[a\rangle\rangle M'_1$ for some input a, then either $M[a\rangle\rangle M_1$, in which case the uniquely defined M'_1 and M_1 match by a), or the input is not specified; but then the environment is not supposed to supply it, such that we can ignore this potential behaviour of C. [10] shows a simple example of an intuitively correct decomposition where due to this feature our correctness holds, while language equivalence, as e.g. required in [5], fails.

The requirement in d) could easily be overlooked: if, in state M', C_i on its own could make an output x that is an input of some C_j, but not allowed there, then there simply exists no x-labelled transition enabled under M' due to the definition of $\|$; but x is under the control of C_i, so it might certainly produce this output, so this must be present in C.

4.2 The Decomposition Algorithm

An essential notion of our algorithm is that of an *admissible operation* for the transformation of an STG; in particular, secure contractions will turn out to be admissible. For understanding this subsection, it is enough to know that each admissible operation is a *tc/pd-operation*, i.e. a transition contraction or the deletion of a place (and its incident arcs). We will introduce the further properties below as the exact definition is tuned to the proof and rather technical.

To initialize the algorithm, one has to choose a *feasible partition* of the signals of N, i.e. a family $(In_i, Out_i)_{i\in I}$ for some set I such that the sets Out_i are a partition of Out_N, for each $i \in I$ we have $In_i \subseteq In_N \cup Out_N$ and furthermore:

(C1) If output signals x and y of N are in structural conflict, then $x \in Out_i$ implies $y \in Out_i$.

(C2) If there are $t, t' \in T_N$ with $t^\bullet \cap {}^\bullet t' \neq \emptyset$ and $l_N(t') \in Out_i$ then $l_N(t) \in In_i \cup Out_i$. ($l_N(t)$ *gives concession* to $l_N(t')$.)

For a feasible partition, the *initial decomposition* is $(C_i)_{i\in I}$, where $C_i = (P, T, W, l_i, M_N, In_i, Out_i)$ is a copy of N except for the labelling and the signals; $l_i(t) = l(t)$ if $l(t) \in In_i \cup Out_i$ and $l_i(t) = \lambda$ otherwise.

Now we transform the C_i stepwise by applying repeatedly an admissible operation to one C_i until either no λ-labelled transitions are left, in which case a decomposition has been found, or one of the following failures occurs:

- The C_i transformed last has a structural auto-conflict. Then the last operation was a contraction of some t, and one adds $l(t)$ to In_i and starts the algorithm for this i again from the new initial C_i.
- There are internal transitions in C_i, but no known admissible operation is applicable. In this case, one adds $l(t)$ to In_i for some internal transition t and restarts as above.

The explanation of the first case is as follows. We take the structural auto-conflict as an indication of a dynamic auto-conflict. Such a dynamic auto-conflict shows that t was labelled in N with a signal C_i should better know about, in order to decide which of the two equally labelled transitions to fire. See [10] for a simple example which also shows that we might get different components if contractions are applied in different orders.

In each step, for some C_i either the number of internal transitions goes down or it stays the same and the number of places decreases – since each admissible operation is a tc/pd-operation – or the number of signals (which is bounded by $|In_N \cup Out_N|$) increases in case of a restart. Thus, eventually a decomposition will be found, although it might not be useful if the C_i are too large – in an extreme case, one could equal N except for the labelling. But even then, the result might be useful, e.g. for splitting off arbiters; and in any case, our algorithm works more often than the approaches presented in the literature so far.

4.3 The Correctness Proof

We will now define admissible operations in two steps, one being structural and the other behavioural.

Definition 4. An operation is *pre-admissible* if it is a tc/pd-operation that when applied to an STG without dynamic auto-conflicts and satisfying (a) and (b) below preserves freeness from auto-concurrency and gives an STG satisfying (a) and (b) again:
(a) There is neither a structural input/output nor a structural λ/output conflict.
(b) If t_2 is an output transition and $t_1^\bullet \cap {}^\bullet t_2 \neq \emptyset$, then t_1 is not internal.

The central notion in our correctness proof is a variant of a bisimulation with an angelic treatment of internal transitions; it is needed to describe in what sense the intermediate stages of our algorithm are correct (like a loop invariant). Any internal transition in an initial C_i corresponds to a signal that this component does not 'see'; if we assume that by some angelic intervention such a transition fires if the signal occurs, then the C_i together work as intended, and this sort of behaviour is captured with an angelic bisimulation. For the final C_i, an angelic bisimulation guarantees correctness, since there are no internal transitions. It is highly interesting that this kind of bisimulation is useful even though we do not assume any angelic nondeterminism in our correctness definition.

Definition 5. A collection of components $(C_i)_{i\in I}$ is an *angelically correct decomposition* of a deterministic STG N, if the parallel composition C of the C_i is defined, $In_C \subseteq In_N$, $Out_C \subseteq Out_N$ and there is an angelic bisimulation relation $\mathcal{B}$ between the markings of N and those of C, i.e. satisfying the following:

1. $(M_N, M_C) \in \mathcal{B}$
2. For all $(M, M') \in \mathcal{B}$, we have:
   (a) If $a \in In_N$ and $M[a\rangle\rangle M_1$, then either $a \in In_C$ and $M'[a\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1' or $a \notin In_C$ and $M'[\lambda\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.

(b) If $x \in Out_N$ and $M[x\rangle\rangle M_1$, then $M'[x\rangle\rangle M_1'$ and $(M_1, M_1') \in \mathcal{B}$ for some M_1'.
(c) If $x \in Out_i$ for some $i \in I$ and $M'\big|_{P_i}[x\rangle\rangle$, then some M_1' and M_1 satisfy $M'[x\rangle\rangle M_1'$, $M[x\rangle\rangle M_1$ and $(M_1, M_1') \in \mathcal{B}$.

This looks very much like Definition 3, but here: $[x\rangle\rangle$ in C might involve additional λ-transitions besides an x-labelled transition; in 2(a) internal transitions are allowed to match an input of N that is not one of C; 2(c) is a combination of 3.2(c) and (d) and guarantees a matching only for some M_1' – this is an angelic part of the definition. It is also angelic that we do not require a match for the firing of only internal transitions in C.

Definition 6. A pre-admissible operation applied to some member of a family $(C_i)_{i \in I}$ that satisfies (a) and (b) of Def. 4 is *admissible* if it preserves angelic correctness w.r.t. N.

Theorem 7. *The algorithm produces are a correct decomposition of N.*

4.4 Admissible Operations

The following theorem shows that secure contractions can be used in the decomposition algorithm, but also others if they do not introduce auto-concurrency. This can possibly be checked without state exploration by using place invariants, or it holds automatically if each transition label occurs only once as assumed in [5]. So far, we have no practical experience with non-secure contractions.

Theorem 8. *Contractions that preserve freeness of auto-concurrency, i.e. secure contractions in particular, are admissible.*

A place p of an STG S is (structurally) *redundant* (see e.g. [4]) if there is a set of places Q with $p \notin Q$, a valuation $V : Q \cup \{p\} \to \mathbb{N}$ and some $c \in \mathbb{N}_0$ with:
– $V(p)M_S(p) - \sum_{q \in Q} V(q)M_S(q) = c$
– for all transitions t, $V(p)(W(t,p) - W(p,t)) - \sum_{q \in Q} V(q)(W(t,q) - W(q,t)) \geq 0$
– for all transitions t, $V(p)W(p,t) - \sum_{q \in Q} V(q)W(q,t) \leq c$

If the third item holds (at least) for all output transitions t, we call *p output-redundant.*

Clearly, the deletion of a redundant place in S turns each reachable marking of S into one of the transformed STG that enables the same transitions, hence the deletion gives a bisimilar STG. Still, it is not completely obvious that such deletions are admissible operations, since the latter are defined w.r.t. the structure of STGs, which certainly changes, and since such a deletion can increase the concurrency.

Theorem 9. *Deleting an (output-)redundant place is an admissible operation.*

A special case of a redundant place is a *loop-only place*, i.e. a marked place p where ${}^\bullet p = p^\bullet$ and all arcs from p have weight 1.

5 Examples

In the realistic examples below, labels have the form $a+$ and $a-$ (denoting rising and falling edges of the signal a). To fit them into the approach presented here, one can e.g. regard the $+$ and $-$ as comments and the a-labelled transitions as toggle-transitions that let rising and falling edges of a alternate.

With the benchmark examples in the table below[2] (circulating in the STG-community), we demonstrate the possible savings w.r.t. the number of reachable markings – by listing the name, the number for this STG, and the numbers for the components of the best decomposition we have found so far. Examples 1-4 are so-called *marked graphs*, where the arc weights are 1 and each place has exactly one transition in its preset and one in its postset; thus, there are no conflicts. E.g. FIFO is the specification of a FIFO-queue controller that has already been studied and decomposed somewhat informally in [3]. Even such a simple case as FIFO *cannot* be decomposed with the methods given in the literature so far, since the deletion of loop-only places is necessary (as in 4) and since during the construction places with 2 or more tokens arise; the latter is no problem for us since we do not restrict ourselves to the treatment of safe nets.

no	name	reach. markings	for the components
1	FIFO	832	26 12 12 8 4 4
2	nak-pa	58	16 13 12
3	adfast	44	21 12
4	mread-8932	8932	36 36 18 18 12 10
5	pe-rcv-ifc	65	53 25
6	tsend_csm	35	25 16
7	mux2	99	63 33
8	locked2	166	34 20 20

For marked graphs where each transition can fire under some reachable marking, secure contractions and deletion of loop-only places always succeed to remove all internal transitions without a restart. Furthermore, if we can find an initial decomposition where each component has at least one λ-transition, then each component will have fewer reachable markings than N in the end.

Examples 5-8 are free-choice nets, each of them having some labels occurring more than once; hence, the methods of [5,9] are not applicable. Restarts were needed in 5, 6 and 8. Globally irrelevant inputs were found in 7 and 8.

Our final example demonstrates that our algorithm can deal with arbitration, splitting off a component one can find in a library. Figure 2 shows on the left the STG specification of a low latency arbiter known e.g. from [11]. Upon a request $R1+$ or $R2+$ (note that there is an unsafe place), the circuit arbitrates with $A1+$ or $A2+$. Critical race behaviour as between $A1+$ or $A2+$ cannot be correctly implemented by a pure logic circuit; additional analogue circuitry is needed to avoid anomalous behaviour like metastability. Present day STG-based

[2] where 6-8 have been adapted slightly for our approach

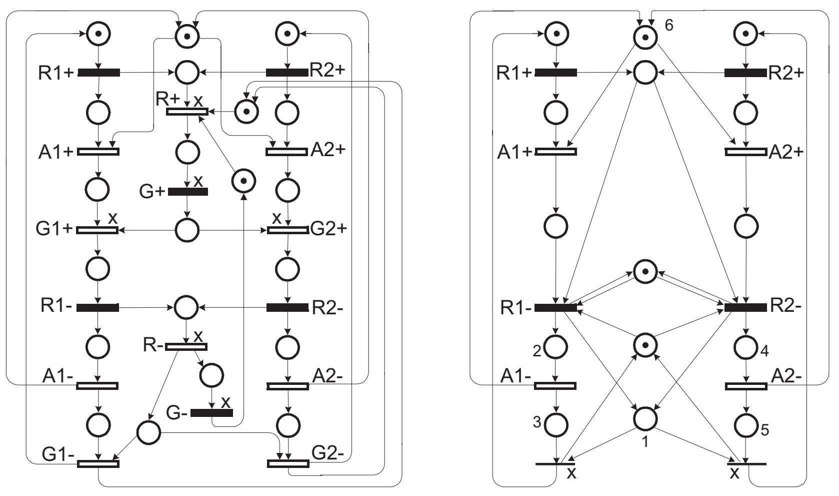


Fig. 2.

tools, therefore, cannot handle such behaviour. The problem is usually solved by using specific library elements, in particular a so-called two-way mutual exclusion (ME) element (e.g. [11]); our decomposition method can give support here – with results that we have proven to be correct.

A component for the generation of the $Ai*$, $i = 1, 2$ and $* \in \{+, -\}$, must have the $Ri*$ as inputs according to the definition of a feasible partition. Applying secure contractions to the x-marked transitions in the corresponding initial C_1 gives the STG on the right in Fig. 2. The remaining internal transitions do not allow a secure contraction; but place 1 is redundant due to $Q = \{2, 3, 4, 5\}$ with $V \equiv 1$. After its deletion, we can perform two secure contractions; by deleting the remaining places in 'the middle column' except 6 – which are redundant –, we get the standard STG representation of the ME-element (e.g. [11]).

This example is concerned with splitting off a library element and not with savings in the state space. In fact, the second component that generates the other output signals $R*$ and $Gi*$, $i = 1, 2$ and $* \in \{+, -\}$, is not smaller than N. But it does not specify critical races (because the $Ai*$ are inputs for this component), and can therefore be implemented with the known STG-based methods.

6 Conclusion and Future Work

Our correctness proof for a quite general STG decomposition algorithm combines structural reasoning with a novel semantic concept (angelic correctness). We have demonstrated the usefulness of our algorithm with some practical examples.

The first open problem concerns determinism: one would get much more freedom in finding components, if one allowed internal signals for the communication between components; then the composition C would not be deterministic anymore with impact on the correctness definition. Also for N, it can be useful to allow nondeterminism: e.g. to treat arbitration in general, it can be necessary to have several enabled transitions with the same label.

Of course, we have to study more examples; for this, integration of the decomposition algorithm into a tool is needed, and this is already under way. Depending on the choice of a feasible start partition and on the choices of signals for restarts, we can arrive at different decompositions. We want to study methods to find good decompositions for various quality criteria (like the overall or the maximal number of reachable markings for the components).

Finally, we want to generalize our correctness criterion to take concurrency into consideration, where it seems to be natural to require the modular implementation to exhibit at least the concurrency prescribed in the specification.

References

1. C. André. Structural transformations giving B-equivalent PT-nets. In Pagnoni and Rozenberg, editors, *Applications and Theory of Petri Nets*, Informatik-Fachber. 66, 14–28. Springer, 1983.
2. J. Beister, G. Eckstein, and R. Wollowski. Cascade: a tool kernel supporting a comprehensive design method for asynchronous controllers. In M. Nielsen, editor, *Applications and Theory of Petri Nets 2000*, LNCS 1825, 445–454. Springer, 2000.
3. J. Beister and R. Wollowski. Controller implementation by communicating asynchronous sequential circuits generated from a Petri net specification of required behaviour. In G. Caucier and J. Trilhe, editors, *Synthesis for Control Dominated Circuits*, 103–115. Elsevier Sci. Pub. 1993.
4. G. Berthelot. Transformations and decompositions of nets. In W. Brauer et al., editors, *Petri Nets: Central Models and Their Properties*, Lect. Notes Comp. Sci. 254, 359–376. Springer, 1987.
5. T.-A. Chu. *Synthesis of Self-Timed VLSI Circuits from Graph-Theoretic Specifications.* PhD thesis, MIT, 1987. Extended abstract in IEEE Int. Conf. Computer Design ICCD '87, 1987, p.220–223.
6. J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Trans. Inform. and Systems*, E80-D, 3:315–325, 1997.
7. D. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent circuits.* MIT Press, Cambridge, 1988.
8. J. Ebergen. Arbiters: an exercise in specifying and decomposing asynchronously communicating components. *Sci. of Computer Programming*, 18:223–245, 1992.
9. A. Kondratyev, M. Kishinevsky, and A. Taubin. Synthesis method in self-timed design. Decompositional approach. In *IEEE Int. Conf. VLSI and CAD*, pages 324–327, 1993.
10. W. Vogler and R. Wollowski. Decomposition in asynchronous circuit design. Technical Report 2002-5, University of Augsburg, http://www.Informatik.Uni-Augsburg.DE/skripts/techreports/, 2002.
11. A. Yakovlev, M. Kishinevsky, A. Kondratyev, and L. Lavagno. Or causality: Modelling and hardware implementation. In R. Valette, editor, *Applications and Theory of Petri Nets 1994*, Lect. Notes Comp. Sci. 815, 568–587. Springer, 1994.

Queue Layouts, Tree-Width, and Three-Dimensional Graph Drawing*

David R. Wood

School of Computer Science
Carleton University
Ottawa, Canada
davidw@scs.carleton.ca

Abstract. A *three-dimensional* (*straight-line grid*) *drawing* of a graph represents the vertices by points in $\mathbb{Z}^3$ and the edges by non-crossing line segments. This research is motivated by the following open problem due to Felsner, Liotta, and Wismath [Graph Drawing '01, *Lecture Notes in Comput. Sci.*, 2002]: *does every n-vertex planar graph have a three-dimensional drawing with $O(n)$ volume*? We prove that this question is almost equivalent to an existing one-dimensional graph layout problem. A *queue layout* consists of a linear order σ of the vertices of a graph, and a partition of the edges into queues, such that no two edges in the same queue are nested with respect to σ. The minimum number of queues in a queue layout of a graph is its *queue-number*. Let G be an n-vertex member of a proper minor-closed family of graphs (such as a planar graph). We prove that G has a $O(1) \times O(1) \times O(n)$ drawing if and only if G has $O(1)$ queue-number. Thus the above question is almost equivalent to an open problem of Heath, Leighton, and Rosenberg [*SIAM J. Discrete Math.*, 1992], who ask whether every planar graph has $O(1)$ queue-number? We also present partial solutions to an open problem of Ganley and Heath [*Discrete Appl. Math.*, 2001], who ask whether graphs of bounded tree-width have bounded queue-number? We prove that graphs with bounded path-width, or both bounded tree-width and bounded maximum degree, have bounded queue-number. As a corollary we obtain three-dimensional drawings with optimal $O(n)$ volume, for series-parallel graphs, and graphs with both bounded tree-width and bounded maximum degree.

1 Introduction

A celebrated result independently due to de Fraysseix, Pach, and Pollack [6] and Schnyder [27] states that every n-vertex planar graph has a (two-dimensional) straight-line grid drawing with $O(n^2)$ area. Motivated by applications in information visualisation, VLSI circuit design and software engineering, there is a growing body of research in three-dimensional graph drawing (see [12] for example). One might expect that in three dimensions, planar graphs would admit straight-line grid drawings with $o(n^2)$ volume. However, this question has remained an

* Research supported by NSERC.

elusive open problem. The main contribution of this paper is to prove that this question of three-dimensional graph drawing is almost equivalent to an existing one-dimensional graph layout problem regarding queue layouts. Furthermore, we establish new relationships between queue-number, tree-width and path-width; and obtain $O(n)$ volume three-dimensional drawings of series-parallel graphs, and graphs with both bounded tree-width and bounded degree.

1.1 Definitions and Notation

Throughout this paper, all graphs G are undirected, simple, connected, and finite with vertex set $V(G)$ and edge set $E(G)$. The number of vertices and maximum degree of G are respectively denoted by $n = |V(G)|$ and $\Delta(G)$. For all disjoint subsets $A, B \subseteq V(G)$, the bipartite subgraph of G with vertex set $A \cup B$ and edge set $\{vw \in E(G) : v \in A, w \in B\}$ is denoted by $G[A, B]$.

A *tree-decomposition* of a graph G consists of a tree T and a collection $\{T_x : x \in V(T)\}$ of subsets T_x (called *bags*) of $V(G)$ indexed by the nodes of T such that:

- $\displaystyle\bigcup_{x \in V(T)} T_x = V(G)$,
- $\forall$ edges $vw \in E(G)$, $\exists$ node $x \in V(T)$ such that $\{v, w\} \subseteq T_x$, and
- $\forall$ nodes $x, y, z \in V(T)$, if y is on the xz-path in T, then $T_x \cap T_z \subseteq T_y$.

The *width* of a tree-decomposition is one less than the maximum size of a bag. A *path-decomposition* is a tree-decomposition where the tree T is a path. The *path-width* (respectively, *tree-width*) of a graph G, denoted by $\mathsf{pw}(G)$ ($\mathsf{tw}(G)$), is the minimum width of a path- (tree-) decomposition of G.

1.2 Three-Dimensional Straight-Line Grid Drawing

A *three-dimensional straight-line grid drawing* of a graph, henceforth called a *three-dimensional drawing*, represents the vertices by distinct points in $\mathbb{Z}^3$, and represents each edge as a line-segment between its end-vertices, such that edges only intersect at common end-vertices. In contrast to the case in the plane, it is well known that every graph has a three-dimensional drawing. We therefore are interested in optimising certain measures of the aesthetic quality of a drawing. If a three-dimensional drawing is contained in an axis-aligned box with side lengths $X - 1$, $Y - 1$ and $Z - 1$, then we speak of an $X \times Y \times Z$ drawing with *volume* $X \cdot Y \cdot Z$. We study three-dimensional drawings with small volume.

Cohen, Eades, Lin, and Ruskey [5] proved that every graph has a three-dimensional drawing with $O(n^3)$ volume, and this bound is asymptotically tight for the complete graph K_n. Calamoneri and Sterbini [4] proved that every 4-colourable graph has a three-dimensional drawing with $O(n^2)$ volume. Generalising this result, Pach, Thiele, and Tóth [23] proved that every k-colourable graph, for fixed $k \geq 2$, has a three-dimensional drawing with $O(n^2)$ volume, and that this bound is asymptotically optimal for the complete bipartite graph with equal sized bipartitions. The first linear volume bound was established by Felsner, Wismath, and Liotta [14], who proved that every outerplanar graph has a drawing with $O(n)$ volume. Poranen [25] proved that series-parallel digraphs have upward three-dimensional drawings with $O(n^3)$ volume, and that this bound can be improved to $O(n^2)$ and $O(n)$ in certain special cases. di Giacomo, Liotta, and Wismath [7] proved that series-parallel graphs with maximum degree three have three-dimensional drawings with $O(n)$ volume. Dujmović, Morin, and Wood [12] proved that every graph G has a three-dimensional drawing with $O(n \cdot \mathsf{pw}(G)^2)$ volume. This implies $O(n \log^2 n)$ volume drawings for graphs of bounded tree-width, such as series-parallel graphs.

Since a planar graph G is 4-colourable and has $\mathsf{pw}(G) \in O(\sqrt{n})$, by the above results of Calamoneri and Sterbini [4], Pach *et al.* [23], and Dujmović *et al.* [12], every planar graph has a three-dimensional drawing with $O(n^2)$ volume. This result also follows from the classical algorithms of de Fraysseix *et al.* [6] and Schnyder [27] for producing plane grid drawings. This paper is motivated by the following open problem due to Felsner *et al.* [14].

Open Problem 1 ([14]). Does every planar graph have a three-dimensional drawing with $O(n)$ volume? In fact, any $o(n^2)$ bound would be of interest.

In this paper we prove that Open Problem 1 is almost equivalent to an existing open problem in the theory of queue layouts.

1.3 Queue Layouts

For a graph G, a linear order of $V(G)$ is called a *vertex-ordering* of G. A *queue layout* of G consists of a vertex-ordering σ of G, and a partition of $E(G)$ into *queues*, such that no two edges in the same queue are *nested* with respect to σ. That is, there are no edges vw and xy in a single queue with $v <_\sigma x <_\sigma y <_\sigma w$. The minimum number of queues in a queue layout of G is called the *queue-number* of G, and is denoted by $\mathsf{qn}(G)$. A similar concept is that of a *stack layout* (or *book embedding*), which consists of a vertex-ordering of G, and a partition of $E(G)$ into *stacks* (or *pages*) such that there are no edges vw and xy in a single stack with $v <_\sigma x <_\sigma w <_\sigma y$. The minimum number of stacks in a stack layout of G is called the *stack-number* (or *page-number*) of G, and is denoted by $\mathsf{sn}(G)$.

Motivated by applications in VLSI layout, fault-tolerant processing, parallel processing, matrix computations, and sorting networks, queue layouts have been extensively studied [19, 20, 24, 26, 29]. Heath and Rosenberg [20] characterised graphs admitting 1-queue layouts as the 'arched leveled planar' graphs, and proved that it is NP-complete to recognise such graphs. This result is in contrast to the situation for stack layouts — the graphs admitting 1-stack layouts are precisely the outerplanar graphs, which can be recognised in polynomial time. On the other hand, it is NP-hard to minimise the number of stacks in a stack layout which respects a given vertex-ordering [17]. However the analogous problem for queue layouts can be solved as follows. As illustrated in Fig. 1, a *k-rainbow*

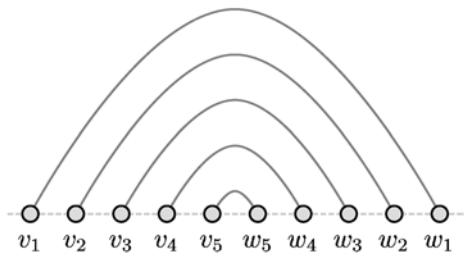


Fig. 1. A rainbow of five edges in a vertex-ordering.

in a vertex-ordering σ consists of a matching $\{v_iw_i : 1 \leq i \leq k\}$ such that $v_1 <_\sigma v_2 <_\sigma \cdots <_\sigma v_k <_\sigma w_k <_\sigma w_{k-1} <_\sigma \cdots <_\sigma w_1$.

A vertex-ordering containing a k-rainbow needs at least k queues. A straightforward application of Dilworth's Theorem [9] proves the converse. That is, a fixed vertex-ordering admits a k-queue layout where k is the size of the largest rainbow. (Heath and Rosenberg [20] describe an $O(m \log\log n)$ time algorithm to compute the queue assignment.) Thus determining $\mathsf{qn}(G)$ can be viewed as the following vertex layout problem.

Lemma 1 ([20]). *The queue-number $\mathsf{qn}(G)$ of a graph G is the minimum, taken over all vertex-orderings σ of G, of the maximum size of a rainbow in σ.*

The relationship between tree-width and stack and queue layouts has previously been studied in [16, 26]. Rengarajan and Veni Madhavan [26] prove that a graph of tree-width at most two (that is, a graph with series-parallel biconnected components [2]) has a 2-stack layout and a 3-queue layout. In the special case of an outerplanar graph a 2-queue layout is constructed. More generally, Ganley and Heath [16] prove that the stack-number $\mathsf{sn}(G) \leq \mathsf{tw}(G)+1$, and ask whether a similar relationship holds for the queue-number.

Open Problem 2 ([16]). Does every graph of bounded tree-width have bounded queue-number?

2 Our Results

This paper contributes the following two theorems. The first, proved in Section 3, provides a partial answer to Open Problem 2.

Theorem 1. *The following classes of graphs have bounded queue-number:*
(1) graphs of bounded path-width, and
(2) graphs of bounded tree-width and bounded maximum degree.
In particular, $\mathsf{qn}(G) \leq \mathsf{pw}(G)$ and $\mathsf{qn}(G) \leq 36\,\mathsf{tw}(G)\Delta(G)$ for every graph G.

A similar upper bound to (1) is obtained by Heath and Rosenberg [20], who show that every graph G has $\mathsf{qn}(G) \leq \lceil \frac{1}{2}\mathsf{bw}(G)\rceil$, where $\mathsf{bw}(G)$ is the bandwidth of G. In many cases this result is weaker than (1) since $\mathsf{pw}(G) \leq \mathsf{bw}(G)$ (see

[8]). Note that since $\mathsf{pw}(G) \in O(\mathsf{tw}(G) \cdot \log n)$ [2], the queue-number $\mathsf{qn}(G) \in O(\mathsf{tw}(G) \cdot \log n)$.

Theorem 2 below relates the volume of a three-dimensional drawing of a graph to its queue-number, and is proved in Section 4. While our motivation is for three-dimensional drawings of planar graphs, the theorem applies to any *proper minor-closed* family of graphs; that is, a graph family which is not the class of all graphs, and is closed under edge-contraction, edge-deletion, and deleting isolated vertices.

Theorem 2. *Let $\mathcal{G}$ be a proper minor-closed family of graphs, and let $F(n)$ be a set of functions closed under taking polynomials (for example, $O(1)$ or $O(\text{polylog}\, n)$). For every graph $G \in \mathcal{G}$, G has a $F(n) \times F(n) \times O(n)$ drawing if and only if G has queue number $\mathsf{qn}(G) \in F(n)$.*

Graphs with constant queue-number include de Bruijn graphs, FFT and Beneš network graphs [20]. By the above-mentioned result of Rengarajan and Veni Madhavan [26], and since graphs with tree-width at most some constant form a proper minor-closed family, Theorems 1 and 2 together imply the following. Part (2) is proved without using queue layouts in [12].

Corollary 1. *The following graphs have three-dimensional drawings with $O(n)$ volume:*
(1) de Bruijn graphs, FFT and Beneš network graphs,
(2) graphs of bounded path-width [12],
(3) graphs of tree-width at most two (series-parallel graphs), and
(4) graphs of bounded tree-width and bounded maximum degree.

Corollary 1 improves and/or generalises the above-mentioned results for three-dimensional drawings of outerplanar graphs, series-parallel graphs, and graphs of bounded tree-width in [7, 12, 14, 25]. Note that the algorithm by Felsner *et al.* [14] closely parallels the construction of 2-queue layouts of outerplanar graphs due to Rengarajan and Veni Madhavan [26], both of which are based on breadth-first search, as is one of our proofs to follows.

3 Queue Layouts and Tree-Width

In this section we prove Theorem 1. Consider a vertex-ordering σ of a graph G. The *vertex cut* in σ at a vertex $v \in V(G)$ is defined to be $|\{x \in V(G) : \exists xy \in E(G),\, x \leq_\sigma v <_\sigma y\}|$. The *vertex separation number* of G is the minimum, taken over all vertex-orderings σ of G, of a maximum vertex cut in σ. A k-rainbow in σ implies σ has a vertex cut of size k. Thus the queue-number of a graph is at most its vertex separation number by Lemma 1. The next result immediately follows, since the vertex separation number of a graph equals its path-width (see [8]).

Lemma 2. *Graphs of bounded path-width have bounded queue-number. In particular, $\mathsf{qn}(G) \leq \mathsf{pw}(G)$ for every graph G.*

To establish our next result we employ a structure called a tree-partition [3, 10, 11, 18, 28]. Let G be a graph, let T be a tree, and let $\{T_x : x \in V(T)\}$ be a partition of $V(G)$ into sets (called *bags*) indexed by the nodes of T. We denote the bag containing a vertex $v \in V(G)$ by $T_{\alpha(v)}$. The pair $(T, \{T_x\})$ is a *tree-partition* of G if for every edge $vw \in E(G)$, either $\alpha(v) = \alpha(w)$ or $\alpha(v)\alpha(w) \in E(T)$. We call vw an *intra-bag* edge if $\alpha(v) = \alpha(w)$ and an *inter-bag* edge otherwise. The *width* of the tree-partition is the maximum size of a bag T_x. The *tree-partition-width* of a graph G, denoted by $\mathsf{tpw}(G)$, is the minimum width of a tree-partition of G. Note that tree-partition-width has also been called *strong tree-width* [3, 28].

Lemma 3. *Graphs of bounded tree-partition-width, which includes graphs of bounded tree-width and bounded maximum degree, have bounded queue-number. In particular,* $\mathsf{qn}(G) \le \frac{3}{2}\mathsf{tpw}(G) \le 36\,\mathsf{tw}(G)\Delta(G)$ *for every graph* G*.*

Proof. Let $(T, \{T_x\})$ be a tree-partition of G with width $\mathsf{tpw}(G)$. Let π be a vertex-ordering of T determined by a lexicographical breadth-first-search of T starting from an arbitrary root node. Then no two edges of T are nested in π. (This is why trees have queue-number one.) Also observe that each node $x \in V(T)$ has at most one incident edge xy with $y <_\pi x$.

Let σ be a vertex-ordering of G such that $v <_\sigma w$ implies $\alpha(v) \le_\pi \alpha(w)$. Suppose edges e_1 and e_2 of G are nested in σ. If e_1 and e_2 are both intra-bag edges then their end-vertices are all in a common bag. Thus there are at most $\frac{1}{2}\mathsf{tpw}(G)$ intra-bag edges in a rainbow of σ. If e_1 and e_2 are both inter-bag edges then the left end-vertex of e_1 and the left end-vertex of e_2 are in a common bag. Thus there are at most $\mathsf{tpw}(G)$ inter-bag edges in a rainbow of σ. Therefore a rainbow in σ can have at most $\frac{3}{2}\mathsf{tpw}(G)$ edges.

The result follows from Lemma 1, and since Ding and Oporowski [10] proved that $\mathsf{tpw}(G) \le 24\,\mathsf{tw}(G)\Delta(G)$ for every graph G. □

Lemmata 2 and 3 establish Theorem 1.

4 Queue Layouts and Three-Dimensional Drawings

In this section we prove Theorem 2. Our proof depends on the following structure introduced by Dujmović *et al.* [12]. An *ordered k-layering* of a graph G consists of a partition $V_1, V_2, \dots, V_k$ of $V(G)$ into *layers*, and a total order $<_i$ of each V_i, such that for every edge vw, if $v <_i w$ then there is no vertex x with $v <_i x <_i w$. The *span* of an edge vw is $|i - j|$ where $v \in V_i$ and $w \in V_j$. An *intra-layer* edge is an edge with zero span. An *X-crossing* consists of two edges vw and xy such that for distinct layers i and j, $v <_i x$ and $y <_j w$. Dujmović *et al.* [12] proved the following (see Fig. 2).

Lemma 4 ([12]). *Let $F(n)$ be a set of functions closed under taking polynomials. Then a graph G has a $F(n) \times F(n) \times O(n)$ drawing if and only if G has an ordered k-layering with no X-crossing, for some $k \in F(n)$. Furthermore, if G has an ordered layering with no X-crossing and maximum edge span s then G has a $O(s) \times O(s) \times O(n)$ drawing.*

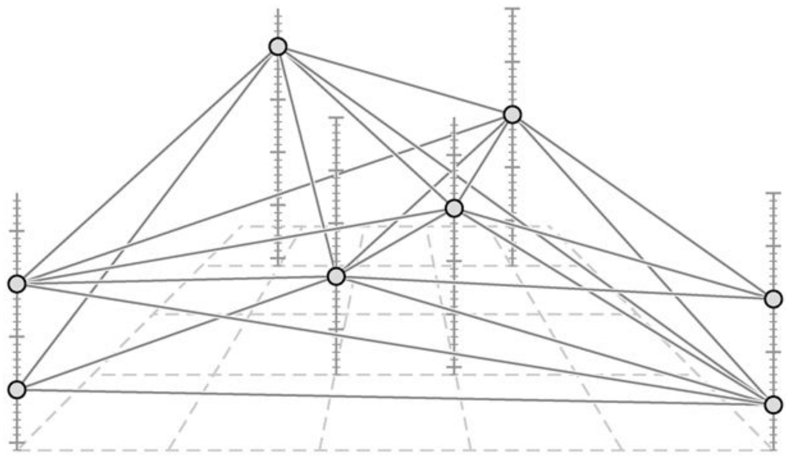


Fig. 2. A three-dimensional drawing produced from an ordered layering with no X-crossing; vertices in each layer are placed on a vertical 'rod'.

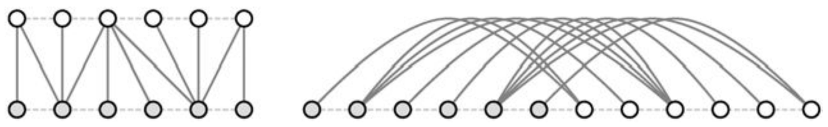


Fig. 3. An ordered 2-layering and a 1-queue layout of a bipartite graph.

Dujmović *et al.* [12] proved that a graph G has an ordered $(\mathsf{pw}(G)+1)$-layering with no X-crossing. That G has a three-dimensional drawing with $O(n \cdot \mathsf{pw}(G)^2)$ volume follows from Lemma 4. A result of Felsner *et al.* [14] also fits into this framework. To construct three-dimensional drawings of outerplanar graphs with $O(n)$ volume, they proved that such a graph has an ordered layering with no X-crossing and maximum edge span at most one. Note that the plane grid graph, which has $\Theta(\sqrt{n})$ path-width and tree-width, has an obvious ordered layering with no X-crossing and maximum edge span one. The 'nested triangles' graph which provides an $\Omega(n^2)$ lower bound on the area of plane grid drawings [6], has an ordered 3-layering with no X-crossing. Thus both of these important examples of planar graphs have three-dimensional drawings with $O(n)$ volume.

Lemma 4 implies that Theorem 2 can be proved if we show that $\mathsf{qn}(G) \in F(n)$ if and only if G has an ordered k-layering with no X-crossing, for some $k \in F(n)$. The next lemma highlights the inherent relationship between ordered layerings and queue layouts. Its proof follows immediately from the definitions (see Fig. 3).

Lemma 5. *A bipartite graph $G = (A, B; E)$ has an ordered 2-layering with no X-crossing and no intra-layer edges if and only if G has a 1-queue layout such that in the corresponding vertex-ordering, the vertices in A appear before the vertices in B.*

We now show that a queue layout can be obtained from an ordered layering with no X-crossing. This result can be viewed as a generalisation of the

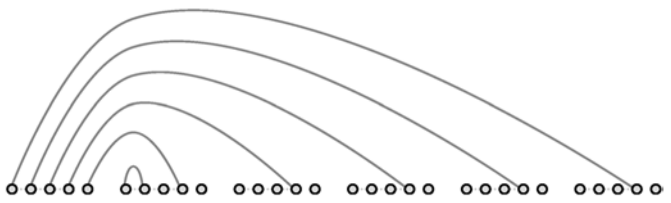


Fig. 4. Maximum rainbow in a vertex-ordering from an ordered layering.

construction of a 2-queue layout of an outerplanar graph by Rengarajan and Veni Madhavan [26] (with $s = 1$).

Lemma 6. *Let G be a graph with an ordered k-layering $\{(V_i, <_i) : 1 \leq i \leq k\}$ with no X-crossing and maximum edge span s. Then $\mathsf{qn}(G) \leq s+1$, and if there are no intra-layer edges then $\mathsf{qn}(G) \leq s$.*

Proof. Let $\sigma = V_1, \dots, V_k$, with each V_i ordered by $<_i$. Let R be the largest rainbow in σ. By Lemma 5, between each pair of layers there is at most one edge in R. A simple inductive argument shows that there is at most s non-intra-layer edges in R; see Fig. 4. No two intra-layer edges are nested in σ. Thus R has at most $s+1$ edges. By Lemma 1, $\mathsf{qn}(G) \leq s+1$. If there are no intra-layer edges then R has at most s edges and $\mathsf{qn}(G) \leq s$. □

We now prove a converse result to Lemma 6. Consider an ordered k-layering with no X-crossing and no intra-layer edges. It is easily seen that the subgraph induced by two layers is a forest of caterpillars. A slightly smaller family of graphs is a forest of stars. A proper vertex-colouring of a graph is called a *star colouring* if each bichromatic subgraph is a forest of stars; that is, every path on four vertices receives at least three distinct colours. The minimum number of colours in a star colouring of a graph G is called the *star chromatic number* of G, and is denoted by $\chi_{\text{st}}(G)$. Nešetřil and Ossona de Mendez [22] proved that every planar graph G has $\chi_{\text{st}}(G) \leq 30$. Many other graph families have bounded star chromatic number, including graphs with bounded maximum degree [1], and graphs with bounded tree-width [15]. In particular, Fertin *et al.* [15] proved that $\chi_{\text{st}}(G) \leq \frac{1}{2}\mathsf{tw}(G)(\mathsf{tw}(G)+3)+1$. More generally, Nešetřil and Ossona de Mendez [22] proved that G has bounded star chromatic number if and only if G is a member of a proper minor-closed family of graphs. In this case, $\chi_{\text{st}}(G)$ is at most a quadratic function of the maximum chromatic number of a minor of G.

Lemma 7. *Let G be a graph with star chromatic number $\chi_{\text{st}}(G) \leq c$, and queue-number $\mathsf{qn}(G) \leq q$. Then G has an ordered t-layering with no X-crossing where*

$$t \ \leq \ c\big(2(c-1)q+1\big)^{c-1} \ .$$

Proof. Let $V_1, \dots, V_c$ be the colour classes of a star colouring of G. Pemmaraju [24] proved that a q-queue graph layout can be 'separated' by a vertex c-colouring to produce a $2(c-1)q$-queue layout with the vertices in each colour class consecutive in the vertex-ordering. (The proof is a straightforward application of

Lemma 1.) Applying this result to the given queue layout and star colouring, we obtain a q'-queue layout of G with vertex-ordering $\sigma = V_1, \dots, V_c$, where $q' = 2(c-1)q$.

For every vertex $v \in V_i$, $1 \leq i \leq c$, and $j \in \{1, \dots, c\} \setminus \{i\}$, let $d_j(v)$ be the degree of v in $G[V_i, V_j]$. Define the jth *label* of v, denoted by $\phi_j(v)$, as follows. If $d_j(v) \geq 2$ then let $\phi_j(v) =$ 'r' (v is the root of a star in $G[V_i, V_j]$). If $d_j(v) = 1$ then let $\phi_j(v)$ be the queue containing the edge in $G[V_i, V_j]$ incident to v. If $d_j(v) = 0$ then let $\phi_j(v)$ be some arbitrary queue. Let the *label* of $v \in V_i$ be $\phi(v) = (\phi_1(v), \dots, \phi_{i-1}(v), \phi_{i+1}(v), \dots, \phi_c(v))$. Let S_i be the set of possible labels for a vertex in V_i. Then $|S_i| = (q'+1)^{c-1}$.

Now group the vertices with the same colour and the same label. Let $V_{i,L} = \{v \in V_i : \phi(v) = L\}$ for all labels $L \in S_i$ and $1 \leq i \leq c$, and consider each $V_{i,L}$ to be ordered by σ. Thus $\{V_{i,L} : 1 \leq i \leq c, L \in S_i\}$ is an ordered layering of G. We denote the jth label of $L \in S_i$ by $L[j]$.

Consider a subgraph $G[V_{i,P}, V_{j,Q}]$ for some $1 \leq i < j \leq c$ and labels $P \in S_i$ and $Q \in S_j$. We claim that all edges in $G[V_{i,P}, V_{j,Q}]$ are in a single queue. If $P[j] =$ 'r' and $Q[i] =$ 'r' then $G[V_{i,P}, V_{j,Q}]$ has no edges. If $P[j] =$ 'r' and $Q[i] = q_a$ for some queue q_a, then all edges in $G[V_{i,P}, V_{j,Q}]$ are in q_a. Similarly, if $Q[i] =$ 'r' and $P[j] = q_a$ for some queue q_a, then all edges in $G[V_{i,P}, V_{j,Q}]$ are in q_a. Finally, consider the case in which $P[j] = q_a$ and $Q[i] = q_b$ for some queues q_a and q_b. If $a \neq b$ then there are no edges in $G[V_{i,P}, V_{j,Q}]$, and if $a = b$ then all edges in $G[V_{i,P}, V_{j,Q}]$ are in queue $q_a (= q_b)$. In each case, all edges in $G[V_{i,P}, V_{j,Q}]$ are in a single queue. By Lemma 5, $V_{i,P}$ and $V_{j,Q}$ form an ordered 2-layering of $G[V_{i,P}, V_{j,Q}]$ with no X-crossing. In general, $\{V_{i,L} : 1 \leq i \leq c, L \in S_i\}$ is an ordered layering of G with no X-crossing. The number of layers is $c(q'+1)^{c-1} = c(2(c-1)q+1)^{c-1}$. □

Lemmata 4, 6 and 7 together with the result of Nešetřil and Ossona de Mendez [22] establish Theorem 2.

5 Conclusion

Theorem 2 implies that a planar graph has a three-dimensional drawing with $O(n)$ volume if it has $O(1)$ queue-number. Thus an affirmative answer to the following open problem due to Heath *et al.* [19] would solve Open Problem 1. In fact, the two problems are almost equivalent. It is possible, however, that a planar graph has non-constant queue-number, yet has say a $O(n^{1/3}) \times O(n^{1/3}) \times O(n^{1/3})$ drawing.

Open Problem 3 ([19, 20]). Does every planar graph have $O(1)$ queue-number?

In 1992, Heath and Rosenberg [20] and Heath *et al.* [19] conjectured that every planar graph *does* have $O(1)$ queue-number. More recently, Pemmaraju [24] provided 'evidence' that the planar graph obtained by repeated stellation of K_3 (that is, by adding a degree three vertex to every face) has non-constant queue-number. This graph does have $O(\log n)$ queue-number [24]. Pemmaraju [24] and Heath [private communication, 2002] conjecture that every planar graph has $O(\log n)$ queue-number. By Theorem 2, this would imply that every planar graph has a three-dimensional drawing with $O(n \operatorname{polylog} n)$ volume. Note that if the stellated K_3 graph, which has tree-width three, has non-constant queue-number then Open Problem 2 would also have a negative answer [16].

The best known upper bound on the queue-number of a planar graph is $O(\sqrt{n})$, which follows from Lemma 2 and the fact that the path-width of a planar graph is $O(\sqrt{n})$ (see [2]). This result can also be proved using a variant of the randomised algorithm of Malitz [21] (see [19]), or the derandomised algorithm of Shahrokhi and Shi [29].

As a final word, we estimate the constants in the $O(n)$ volume bound of Corollary 1. Take a graph G with bounded tree-width $\mathsf{tw}(G) \leq k$ and bounded maximum degree $\Delta(G) \leq d$. Then $\chi_{\mathrm{st}}(G) \leq \frac{1}{2}k^2 + o(k^2)$ [15] and $\mathsf{qn}(G) \leq 36kd$ by Lemma 3. By Lemma 7, G has an ordered layering with no X-crossing and approximately $k^2(36k^3d)^{k^2/2}$ layers. By Lemma 4, G has a three-dimensional drawing with approximately $O(k^4(36k^3d)^{k^2} \cdot n)$ volume. As another example, a series-parallel graph G has $\mathsf{tw}(G) \leq 2$ [2], $\mathsf{qn}(G) \leq 3$ [26], and $\chi_{\mathrm{st}}(G) \leq 6$ [15]. By Lemma 7, G has an ordered layering with no X-crossing and at most $6 \cdot 31^5$ layers. By Lemma 4, the constant in the $O(n)$ volume bound of Corollary 1 for series-parallel graphs is at least $36 \cdot 31^{10} \approx 2.9 \times 10^{16}$. It is an interesting open problem to construct linear volume three-dimensional drawings with a smaller constant in the $O(n)$ volume bound.

Acknowledgements

Thanks to Prosenjit Bose, Jurek Czyzowicz, Hubert de Fraysseiz, Patrice Ossona de Mendez, and Pat Morin for helpful discussions.

Note added in proof: Dujmović and Wood [13] recently solved Open Problem 2. That is, graphs of bounded tree-width have bounded queue-number, and hence have three-dimensional drawings with linear volume.

References

[1] N. Alon, C. McDiarmid, and B. Reed, Acyclic coloring of graphs. *Random Structures Algorithms*, **2(3)**:277–288, 1991.

[2] H. L. Bodlaender, A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, **209(1-2)**:1–45, 1998.

[3] H. L. Bodlaender and J. Engelfriet, Domino treewidth. *J. Algorithms*, **24(1)**:94–123, 1997.

[4] T. Calamoneri and A. Sterbini, 3D straight-line grid drawing of 4-colorable graphs. *Inform. Process. Lett.*, **63(2)**:97–102, 1997.

[5] R. F. Cohen, P. Eades, T. Lin, and F. Ruskey, Three-dimensional graph drawing. *Algorithmica*, **17(2)**:199–208, 1996.

[6] H. de Fraysseix, J. Pach, and R. Pollack, How to draw a planar graph on a grid. *Combinatorica*, **10(1)**:41–51, 1990.

[7] E. di Giacomo, G. Liotta, and S. Wismath, Drawing series-parallel graphs on a box. In S. Wismath, ed., *Proc. 14th Canadian Conf. on Computational Geometry (CCCG '02)*, The University of Lethbridge, Canada, 2002.

[8] J. Díaz, J. Petit, and M. Serna, A survey of graph layout problems. *ACM Comput. Surveys*, to appear.

[9] R. P. Dilworth, A decomposition theorem for partially ordered sets. *Ann. of Math. (2)*, **51**:161–166, 1950.

[10] G. Ding and B. Oporowski, Some results on tree decomposition of graphs. *J. Graph Theory*, **20(4)**:481–499, 1995.

[11] G. Ding and B. Oporowski, On tree-partitions of graphs. *Discrete Math.*, **149(1-3)**:45–58, 1996.

[12] V. Dujmović, P. Morin, and D. R. Wood, Path-width and three-dimensional straight-line grid drawings of graphs. In M. Goodrich, ed., *Proc. 10th International Symp. on Graph Drawing (GD '02)*, Lecture Notes in Comput. Sci., Springer, to appear.

[13] V. Dujmović and D. R. Wood, Tree-partitions of k-trees with applications in graph layout. Tech. Rep. TR-02-03, School of Computer Science, Carleton University, Ottawa, Canada, 2002.

[14] S. Felsner, S. Wismath, and G. Liotta, Straight-line drawings on restricted integer grids in two and three dimensions. In P. Mutzel, M. Jünger, and S. Leipert, eds., *Proc. 9th International Symp. on Graph Drawing (GD '01)*, vol. 2265 of *Lecture Notes in Comput. Sci.*, pp. 328–342, Springer, 2002.

[15] G. Fertin, A. Raspaud, and B. Reed, On star coloring of graphs. In A. Brandstädt and V. B. Le, eds., *Proc. 27th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '01)*, vol. 2204 of *Lecture Notes in Comput. Sci.*, pp. 140–153, Springer, 2001.

[16] J. L. Ganley and L. S. Heath, The pagenumber of k-trees is $O(k)$. *Discrete Appl. Math.*, **109(3)**:215–221, 2001.

[17] M. R. Garey, D. S. Johnson, G. L. Miller, and C. H. Papadimitriou, The complexity of coloring circular arcs and chords. *SIAM J. Algebraic Discrete Methods*, **1(2)**:216–227, 1980.

[18] R. Halin, Tree-partitions of infinite graphs. *Discrete Math.*, **97**:203–217, 1991.

[19] L. S. Heath, F. T. Leighton, and A. L. Rosenberg, Comparing queues and stacks as mechanisms for laying out graphs. *SIAM J. Discrete Math.*, **5(3)**:398–412, 1992.

[20] L. S. Heath and A. L. Rosenberg, Laying out graphs using queues. *SIAM J. Comput.*, **21(5)**:927–958, 1992.

[21] S. M. Malitz, Graphs with E edges have pagenumber $O(\sqrt{E})$. *J. Algorithms*, **17(1)**:71–84, 1994.

[22] J. Nešetřil and P. Ossona de Mendez, Colorings and homomorphisms of minor closed classes. Tech. Rep. 2001-025, Institut Teoretické Informatiky, Universita Karlova v Praze, Czech Republic, 2001.

[23] J. Pach, T. Thiele, and G. Tóth, Three-dimensional grid drawings of graphs. In G. Di Battista, ed., *Proc. 5th International Symp. on Graph Drawing (GD '97)*, vol. 1353 of *Lecture Notes in Comput. Sci.*, pp. 47–51, Springer, 1998.

[24] S. V. Pemmaraju, *Exploring the Powers of Stacks and Queues via Graph Layouts*. Ph.D. thesis, Virginia Polytechnic Institute and State University, Virginia, U.S.A., 1992.

[25] T. Poranen, A new algorithm for drawing series-parallel digraphs in 3D. Tech. Rep. A-2000-16, Dept. of Computer and Information Sciences, University of Tampere, Finland, 2000.

[26] S. Rengarajan and C. E. Veni Madhavan, Stack and queue number of 2-trees. In D. Ding-Zhu and L. Ming, eds., *Proc. 1st Annual International Conf. on Computing and Combinatorics (COCOON '95)*, vol. 959 of *Lecture Notes in Comput. Sci.*, pp. 203–212, Springer, 1995.

[27] W. Schnyder, Planar graphs and poset dimension. *Order*, **5(4)**:323–343, 1989.

[28] D. Seese, Tree-partite graphs and the complexity of algorithms. In L. Budach, ed., *Proc. International Conf. on Fundamentals of Computation Theory*, vol. 199 of *Lecture Notes in Comput. Sci.*, pp. 412–421, Springer, 1985.

[29] F. Shahrokhi and W. Shi, On crossing sets, disjoint sets, and pagenumber. *J. Algorithms*, **34(1)**:40–53, 2000.

Author Index