LNCS 4337

S. Arun-Kumar
Naveen Garg (Eds.)

FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science

26th International Conference
Kolkata, India, December 2006
Proceedings

Springer

Lecture Notes in Computer Science 4337

Commenced Publication in 1973
Founding and Former Series Editors:
Gerhard Goos, Juris Hartmanis, and Jan van Leeuwen

S. Arun-Kumar   Naveen Garg (Eds.)

FSTTCS 2006: Foundations of Software Technology and Theoretical Computer Science

26th International Conference
Kolkata, India, December 13-15, 2006
Proceedings

Springer

Volume Editors

S. Arun-Kumar
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi 110016, India
E-mail: sak@cse.iitd.ac.in

Naveen Garg
Department of Computer Science and Engineering
Indian Institute of Technology Delhi
New Delhi 110016, India
E-mail: naveen@cse.iitd.ac.in

Library of Congress Control Number: 2006937157

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

LNCS Sublibrary: SL 1 – Theoretical Computer Science and General Issues

ISSN 0302-9743
ISBN-10 3-540-49994-6 Springer Berlin Heidelberg New York
ISBN-13 978-3-540-49994-7 Springer Berlin Heidelberg New York

Springer is a part of Springer Science+Business Media

springer.com

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India
Printed on acid-free paper SPIN: 11944836 06/3142 5 4 3 2 1 0

Preface

Welcome to the proceedings of the 26th annual conference on the Foundations of Software Technology and Theoretical Computer Science (FSTTCS). FSTTCS is organized by the Indian Association for Research in Computer Science (IARCS) which it helped create by bringing together academic computer scientists from various parts of India. Over the years several changes have taken place in the conference, beginning with the first Springer LNCS edition in 1984. The first three proceedings were printed and published in India by the Tata Institute of Fundamental Research. Since then other changes, such as an international Programme Committee and pre-conference and post-conference workshops, have helped nurture and enhance the status of this conference among computer scientists interested in foundational research.

Along with these changes there are quite a few invariant properties this conference enjoys. It has always been held in India, and always in the second or third week of December, which is a good time to travel to and within the country. It is also the most convenient time to meet Indian researchers, most of whom would be temporarily free of teaching and administrative commitments.

This year for the first time in its history, FSTTCS was held in the historic city of Kolkata (formerly Calcutta) at the Indian Statistical Institute (ISI). FSTTCS is one of the events to commemorate the Platinum jubilee year of ISI and we are happy to be part of this celebration. Subhas C. Nandy of ISI Kolkata took full responsibility as chairman of the Organizing Committee.

The conference attracted 155 submissions with authors from 29 countries. We thank the authors for their interest in this conference. The reputation of a conference is effectively determined by its Programme Committee, the refereeing process and the Invited talks. This year, as in the past, we were able to get highly respected researchers to serve on our Programme Committee. The refereeing process too has been of a very high quality. The Programme Committee deliberated on each of the submitted papers and the accompanying referee reports and finally decided to select 34 submissions as being worthy of publication.

The invited speakers this year were Gordon Plotkin of the University of Edinburgh, UK, Emo Welzl of ETH Zurich, Switzerland, Gérard Boudol of INRIA, Sophia Antipolis, France, David Shmoys of Cornell University, USA and Eugene Asarin of Université Paris 7, France.

In keeping with what has now become tradition, two workshops on recent advances in areas of current interest to the community were also organized. The themes this year were *Timed Systems* (organized by Deepak D'Souza and Supratik Chakraborty) and *Approximation Algorithms* (organized by Anupam Gupta and Amit Kumar).

We would like to thank the authors who submitted papers and the Programme Committee for helping us maintain the standard of this conference. We especially thank the authors of the selected papers which form the bulk of this volume. We would like to thank Springer for agreeing to publish these proceedings in their prestigious *Lecture Notes in Computer Science* series, which has in no small way contributed to the status of this conference in academic circles.

We would also like to thank ISI Kolkata and Subhas Nandy's team for hosting FSTTCS, IIT Delhi and MPI-Informatik for providing support and of course, IARCS, the parent organization that FSTTCS gave birth to!

December 2006

Naveen Garg
S. Arun-Kumar
Programme Chairs
FSTTCS 2006

Organization

FSTTCS 2006 and the associated workshops were held at the Indian Statistical Institute, Kolkata.

Programme Committee

Amit Kumar
(IIT Delhi, India)
Anil Seth
(IIT Kanpur, India)
Anuj Dawar
(Cambridge University, UK)
Anupam Gupta
(CMU, USA)
Ashish Tiwari,
(SRI, USA)
Astrid Kiehn
(IIT Delhi, India)
Dale Miller
(INRIA-Futurs, France)
Deepak D'Souza
(IISc, India)
Edgar Ramos
(UIUC, USA)
Giuseppe Italiano
(Universitá di Roma, Italy)
Helmut Veith
(TU München, Germany)
Javier Esparza
(University of Stuttgart, Germany)
Joost-Pieter Katoen
(RWTH, Germany)
Kavitha Telikepalli
(IISc, India)
Madhavan Mukund
(CMI, India)
Manindra Agrawal
(IIT Kanpur, India)
Marco Pistore
(DIT Trento, Italy)
Marina Papatriantafilou
(Chalmers UT, Sweden)
Naveen Garg
(IIT Delhi, India), Co-chair
Neal Young
(UC Riverside, USA)
Nobuko Yoshida
(Imperial College, UK)
Prakash Panangaden
(McGill University, Canada)
Radha Jagadeesan
(DePaul University, USA)
Rohit Khandekar
(University Waterloo, Canada)
S. Arun-Kumar
(IIT Delhi, India), Co-chair
Sriram K. Rajamani
(Microsoft Research, USA)
Subhas C. Nandy
(ISI Kolkata, India)
Supratik Chakraborty
(IIT Bombay, India)
Susanne Albers
(University of Freiburg, Germany)
Yuval Rabani
(Technion, Israel)

Referees

Abhik Roychoudhury
Abraham Flaxman
Achim Blumensath
Aditya Nori
Amitabha Bagchi
Andrea Pacifici
Andrey Rybalchenko
Andrzej Murawski
Angelika Mader

Antoine Meyer
Arijit Bishnu
Ashish Goel
Ashwin Nayak
Avik Chaudhuri
B. Meenakshi
Barbara Koenig
Benedikt Bollig
Bharat Adsul
Bhargab Bhattacharya
Bimal Roy
Binay K. Bhattacharya
Boris Koldehofe
C. A. Murthy
Carsten Kern
Catuscia Palamidessi
Christian Schallhart
Daniel Hirschkoff
Daniel Willems
Daniele Varacca
David N. Jansen
Dino Distefano
Doina Precup
Elad Schiller
Erika Abraham
Fedor Fomin
Franck van Breugel
G. Sivakumar
G. Paun
Guido Proietti
Herbert Wiklicky
Hoeteck Wee
Holger Schlingloff
Igor Walukiewicz
Irit Katriel
J. Radhakrishnan
Jayalal Sarma M. N.
Jochen Konemann
Johannes Kinder
K. Narayan Kumar
Kamal Lodaya
Kapil Vaswani
Kasturi R. Varadarajan
Katarzyna Paluch
Khaled Elbassioni
Kishan Gupta
Kohei Honda
Konstantin Korovin
Krishna S.
Krishnendu Chatterjee
Krishnendu Mukhopadhya
Kunal Talwar
Kunsoo Park
Leonard Schulman
Lukasz Kowalik
Luke Ong
Madhu Gopinathan
Madhu Sudan
Madhusudan P.
Marco Carbone
Marielle Stoelinga
Marko Samer
Markus Holzer
Martin Berger
Martin Neuhusser
Massimiliano Caramia
Matthias Nickles
Maurizio Naldi
Meena Mahajan
Michael Kohler
Nabil Mustafa
Nicolas Markey
Nir Piterman
Norm Ferns
Oded Regev
Oleg Sokolsky
Olivier Carton
Om P. Damani
P. S. Thiagarajan
Palash Sarkar
Pallab Dasgupta
Paolo Zuliani
Paritosh Pandya
Pavithra Prabhakar
Peter Bro Miltersen
Prakash Chandrasekaran
Pranab Sen
Prasad Naldurg
Prasad Sistla
Pushpak Bhattacharya
R. Ramanujam
Rahul Jain
Ramesh Hariharan
Ran Canetti
Rana Barua
Richard Mayr
Robert Schweller
Rohit Parikh
Ron van der Meyden
Ronald de Wolf
Russ Harmer
Ryan Williams
S. P. Suresh
S. Ramesh
Sanjiva Prasad
Sariel Har-Peled
Satyanarayana Valluri
Saugata Basu
Sergey Bereg
Soeren Laue
Srinath Sridhar
Sriram S.
Stefan Funke
Stefan Rieger
Stephen Kobourov
Sudeb P. Pal
Sudhakar G.
Suman Roy
Sumit Ganguly
Sumit Jha
Surender Baswana
Susmita Sur-Kolay
T. S. Hsu
Thomas Noll
Thomas Thierauf
Tingting Han
Umesh Bellur
Véronique Bruyère
V. Arvind
Valentin Goranko
Vinayaka Pandit
Vinodchandran Variyam
Walter Vogler
Willem Conradie
Yaoyun Shi
Yichen Xie

Sponsoring Institutions

Indian Association for Research in Computer Science
Indian Institute of Technology, Delhi
Indian Statistical Institute, Kolkata

Table of Contents

Invited Papers

Contributed Papers – Track A

Contributed Papers – Track B

Shared-Variable Concurrency: A Proposal
(Abstract)

Gérard Boudol

INRIA Sophia Antipolis

In this talk I discuss the semantics of shared-variable concurrency, aka multithreading. There are two well-known ways of managing concurrent threads: one either uses a *preemptive* or a *cooperative* scheduling discipline. In the former, a program, or more precisely its executable version, can be interrupted at any time during its execution by an external device, the scheduler, and the resources needed for execution are then given to another concurrent component for a while. This is perfect for executing concurrent *processes*, which do not share memory. In this case, the programmer does not have to care about the relative performance of the various processes in the system: this is the task of the scheduler. Unfortunately, it is very difficult to program multi-threaded applications with this model. The main difficulty is with *data races*, that is conflicting concurrent accesses to the memory. Although it is very easy to provide a formal "interleaving" semantics for preemptive multi-threading, this semantics usually does not coincide with what is actually implemented. In particular, the grain of atomicity is generally not preserved by the implementation, and a program may be time-sliced at some points of its execution which make no sense at the user level, and the consequence is that there is no clear semantics for the race conditions (see [14] for instance). It is therefore necessary to complement preemptive multi-threading with elaborate synchronization techniques, that require a real expertise from the programmer to be used (see [3]), and to design methods to analyze concurrent programs in order to make them "thread safe", avoiding or detecting race conditions [1,6,11]. We shall not follow the preemptive approach in our proposal for shared-memory concurrency semantics.

In the *cooperative* programming model, a thread decides, by means of specific instructions (like yield for instance), when to leave its turn to another concurrent thread, and the scheduling is therefore distributed among the components. This model, in which there is no data race, has been advocated as a better model than the preemptive one for programming some modern, massively concurrent applications. However, this model also has its drawbacks, the main one being that if the active thread runs into an error, or raises an uncaught exception, or diverges, then the model is broken, in the sense that no other component will have a chance to execute. In particular, in cooperative programming, we have to avoid divergence in some way. Still, we need to be able to program non-terminating applications. Any server for instance should conceptually have an infinite life duration, and should not be programmed to stop after a while. Such a

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 1–3, 2006.

server should not enter into an infinite loop however, it should rather be infinitely often waiting for a new request. In other words, in cooperative programming, programs *must be cooperative*, or *fair*, that is, they should be guaranteed to either terminate or suspend themselves infinitely often. Since we would also like to be able to reuse sequential code in a multi-threaded context, a challenge is: can we design a concurrency semantics that would be basically cooperative, in order to avoid data races, but where any thread – and in particular purely sequential code – is guaranteed to be fair?

Dealing with a higher-order imperative programming model à la ML for sequential code (some other choices are obviously possible), our proposal to solve the above mentioned challenge is to introduce a touch of preemptive scheduling into the cooperative model. The idea is very simple: it is to consider that every *recursive call* to a function should be regarded as a *suspensive* operation, that yields the scheduler for a while. We think that this is a natural and intuitive idea, since recursion appears to be the source of non-termination. Moreover, this does not introduce any data race. However, we then have to face a technical problem, which is that recursion, or more accurately non-termination, may occur in indirect ways in a ML-like, or, for that matter, a Scheme-like language. Indeed, it is well-known that, on the one hand, recursion can be encoded in the pure λ-calculus, and, on the other hand, that recursion can also be encoded using circular reference (this is indeed the way it is implemented), as shown by Landin long ago [8]. The well-known method to recover from the first difficulty is to use a *type system.* In this talk I show that we can use a type and *effect* system [9] for dealing with the second difficulty.

To preclude circular references, we stratify the memory into *regions*, in such a way that functional values stored in a given region may only have a latent effect, such as reading a reference, in strictly "lower" regions. This allows us to define, by induction on the stratification, a *realizability interpretation* of the types and effects, for which the type and effect system is sound. From this we conclude that typable expressions can only diverge if they perform an infinite number of recursive calls, that is, if they suspend infinitely often, in our mixed cooperative/preemptive model. This termination argument is quite classical and general (see [2,5]): it was first used by Tait in [15], under the name of "convertibility", and then by Girard (see [5]) with his "candidats de réductibilité", and by Tait again [16] who called it the "realizability" technique, since it relies on an interpretation of types closely related to Kleene's original recursive realizability interpretation [7]. The realizability interpretation is also a special case of a "logical relation" [10,13]. As far as I can see, this technique has not been previously used for higher-order imperative languages: the work that is technically the closest to ours, and which was our main source of inspiration, is the one by Pitts and Stark [12], but their logical relation (intended to provide a means to prove observational equivalence of programs, not to prove termination) is restricted to a language where the memory can only contain values of basic types. In the talk I mention another application of typing termination, namely in typing "termination leaks" in information flow control (see [4]).

References

1. M. Abadi, C. Flanagan, S.N. Freund, *Types for safe locking: static race detection for Java,* ACM TOPLAS Vol. 28 No. 2 (2006) 207-255.
2. H. Barendregt, *Lambda Calculi with Types,* in Handbook of Logic in Computer Science, Vol. 2 (S. Abramsky, Dov M. Gabbay & T.S.E. Maibaum, Eds.), Oxford University Press (1992) 117-309.
3. A.D. Birrel, *An introduction to programming with threads,* SRC Report 35, DEC (1989).
4. G. Boudol, *On typing information flow,* Intern. Coll. on Theoretical Aspects of Computing, Lecture Notes in Comput. Sci. 3722 (2005) 366-380.
5. J.-Y. Girard, Y. Lafont, P. Taylor, *Proofs and Types,* Cambridge Tracts in Theoretical Computer Science 7, Cambridge University Press (1989).
6. D. Grossman, *Type-safe multithreading in Cyclone,* TLDI'03 (2003) 13-25.
7. S.C. Kleene, *On the interpretation of intuitionistic number theory,* J. of Symbolic Logic, Vol. 10 (1945) 109-124.
8. P.J. Landin, *The mechanical evaluation of expressions,* Computer Journal Vol. 6 (1964) 308-320.
9. J.M. Lucassen, D.K. Gifford, *Polymorphic effect systems,* POPL'88 (1988) 47-57.
10. J.C. Mitchell, *Foundations for Programming Languages,* MIT Press (1996).
11. P. O'Hearn, *Resources, concurrency and local reasoning,* CONCUR'04, Lecture Notes in Comput. Sci. 3170 (2004) 49-67.
12. A. Pitts, I. Stark, *Operational reasoning for functions with local state,* in Higher-Order Operational Techniques in Semantics (A. Gordon and A. Pitts, Eds), Publications of the Newton Institute, Cambridge Univ. Press (1998) 227-273.
13. G. Plotkin, *Lambda-definability and logical relations,* Memo SAI-RM-4, University of Edinburgh (1973).
14. J.C. Reynolds, *Toward a grainless semantics for shared-variable concurrency,* FST-TCS'04, Lecture Notes in Comput. Sci. 3328 (2004) 35-48.
15. W. Tait, *Intensional interpretations of functionals of finite type I,* J. of Symbolic Logic 32 (1967) 198-212.
16. W. Tait, *A realizability interpretation of the theory of species,* Logic Colloquium, Lecture Notes in Mathematics 453 (1975) 240-251.

Hennessy-Plotkin-Brookes Revisited*

Gordon Plotkin

Laboratory for the Foundations of Computer Science, School of Informatics,
University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, Scotland
gdp@inf.ed.ac.uk

In [3] Hennessy and Plotkin gave a domain-theoretic semantics fully abstract for SIP (the Simple Imperative Programming language) + parallelism together with a certain unnatural synchronisation construct, but not, unfortunately, abstract without that construct. Later, in [1], Brookes gave a fully-abstract trace semantics for a slight variation on SIP + parallelism. In his semantics, meanings of programs are sets of traces subject to certain closure conditions, namely 'stuttering' and 'mumbling'; traces are sequences of pairs of states.

We revisit these results in the light of the recent Plotkin, Power, et al algebraic theory of effects, a development of Moggi's monadic account of computational effects [6,2]. In the algebraic approach one considers semantic domains as free algebras for equational theories of the operations giving rise to the effects at hand, see, for example [5,4]. The original Hennessy-Plotkin semantics can be given by such a theory including a 'computation suspension' operator d; it is also isomorphic to the trace semantics where one does not impose closure conditions. It turns out that one can obtain an improved variant of Brookes' semantics by adding two natural equations for the suspension operator; these correspond in a one-to-one manner with a slight variant of Brookes' closure conditions on traces. The resulting semantics is fully abstract for SIP + parallelism.

References

1. S. D. Brookes, Full Abstraction for a Shared-Variable Parallel Language, *Inf. Comput.*, Vol. 127, No. 2, pp. 145–163, 1996.
2. N. Benton, J. Hughes, & E. Moggi, *Monads and Effects*, in *APPSEM '00 Summer School, 2000*, LNCS, Vol. 2395, pp. 42–122, Springer Verlag, 2000.
3. M. C. B. Hennessy & G. D. Plotkin, Full Abstraction for a Simple Parallel Programming Language, *Proc. MFCS '79*, LNCS, Vol. 74, pp. 108-120, Springer-Verlag, 1979.
4. J. M. E. Hyland, G. D. Plotkin & A. J. Power, Combining Effects: Sum and Tensor, *TCS*, Vol. 357, Nos. 1–3, pp. 70–99, 2006.
5. G. D. Plotkin & A. J. Power, Notions of Computation Determine Monads, *Proc. FOSSACS '02*, LNCS, Vol. 2303, pp. 342–356, Springer-Verlag, 2002.
6. E. Moggi, Computational Lambda-Calculus and Monads, *Proc. LICS '89*, pp. 14–23, IEEE Press, 1989.

* This work has been done with the support of EPSRC grants GR/M56333 and GR/S86372/01 and a Royal Society-Wolfson research merit award.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, p. 4, 2006.

Approximation Algorithms for 2-Stage Stochastic Optimization Problems

Chaitanya Swamy[1,⋆] and David B. Shmoys[2,⋆⋆]

[1] Department of Combinatorics & Optimization, University of Waterloo,
Waterloo, ON N2L 3G1
cswamy@math.uwaterloo.ca

[2] School of ORIE and Department of Computer Science, Cornell University
Ithaca, NY 14853
shmoys@cs.cornell.edu

Abstract. Stochastic optimization is a leading approach to model optimization problems in which there is uncertainty in the input data, whether from measurement noise or an inability to know the future. In this survey, we outline some recent progress in the design of polynomial-time algorithms with performance guarantees on the quality of the solutions found for an important class of stochastic programming problems — 2-stage problems with recourse. In particular, we show that for a number of concrete problems, algorithmic approaches that have been applied for their deterministic analogues are also effective in this more challenging domain. More specifically, this work highlights the role of tools from linear programming, rounding techniques, primal-dual algorithms, and the role of randomization more generally.

1 Introduction

Uncertainty is a facet of many decision environments and might arise due to various reasons, such as unpredictable information revealed in the future, or inherent fluctuations caused by noise. Stochastic optimization provides a means to handle uncertainty by modeling it by a probability distribution over possible realizations of the actual data called scenarios. The field of stochastic optimization or stochastic programming, has its roots in the work of Dantzig [4] and Beale [1] in the 1950s, and has increasingly found application in a wide variety of areas, including transportation models, logistics, financial instruments, and network design.

An important and widely used model in stochastic programming is the *2-stage recourse model*: first, given only distributional information about (some of) the data, one commits on initial (*first-stage*) actions. Then, once the actual data is realized according to the distribution, further *recourse actions* can be taken (in the *second stage*) to augment the earlier solution and satisfy the revealed requirements. The aim is to choose the initial actions so as to minimize the

[⋆] Research supported partially by NSERC grant 327620-06.

[⋆⋆] Research supported partially by NSF grant CCR-0430682.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 5–19, 2006.

expected total cost incurred. Typically the recourse actions entail making decisions in rapid reaction to the observed scenario, and are therefore more costly than decisions made ahead of time. Thus there is a trade-off between committing initially having only imprecise information while incurring a lower cost, and deferring decisions to the second–stage when we know the input precisely but the costs are higher. Many applications come under this setting, and much of the textbook of Birge and Louveaux [2] is devoted to models and algorithms for this class of problems.

A commonly cited example involves a setting where a company has to decide where to set up facilities to serve client demands. Typically the demand pattern is not known precisely at the outset, but one might be able to obtain, through simulation models or surveys, statistical information about the demands. This motivates the following 2-step decision process: in the first-stage, given only distributional information about the demands (and deterministic data for the facility opening costs), one must decide which facilities to open initially. Once the actual input (the client demands) is realized according to this distribution, we can extend the solution by opening more facilities, incurring a recourse cost, and we have to assign the realized demands to open facilities. This is the *2-stage stochastic uncapacitated facility location* problem. The recourse costs are usually higher than the original ones (because opening a facility later would involve deploying resources with a small lead time), could be different for the different facilities, and could even depend on the realized scenario.

The formal model. The 2-stage recourse model can be formalized as follows: we are given a probability distribution over possible realizations of the data called scenarios and we construct a solution in two stages. First, we may take some decisions to construct an anticipatory part of the solution, x, incurring a cost of $c(x)$. Then a scenario A is realized according to the distribution, and in the second-stage we may augment the initial decisions by taking recourse actions y_A, (if necessary) incurring a certain cost $f_A(x, y_A)$. The goal is to choose the initial decisions so as to minimize the expected total cost, $c(x) + \mathrm{E}_A\big[f_A(x, y_A)\big]$, where the expectation is taken over all scenarios according to the given probability distribution.

An important issue that we have left unspecified above is the question of how the scenario-distribution is represented. One simple approach is to assume that we are given as part of the input description a list that explicitly enumerates each scenario (occurring with non-zero probability) and its probability of occurrence. However, this causes a significant blow-up in the input size, since the distribution can easily have support size that is exponential in the other input parameters, that is, the non-stochastic portion of the input; for example, in stochastic facility location, consider the case where the demand of each client is set independently. Thus, to ensure that a "polynomial-time" algorithm in this model has running time polynomial in the other input parameters, one must restrict oneself to distributions with a polynomial-size support, which is a severe restriction; we shall call this the *polynomial-scenario model* to reflect this fact. The distribution mentioned above is captured by the *independent-activation model* introduced by Immorlica et al. [11], where the scenario-distribution is a product of independent distributions (described in the input). Typically, there is an underlying set of elements (clients) and a scenario is generated by independent choices (setting the demands) made for each element. Independent distributions allow one to succinctly specify a class of distributions with exponentially many scenarios, and have been used in the Computer Science community to model uncertainty in various settings [13,18,5]. However, many of the underlying stochastic applications often involve *correlated data* (e.g., in stochastic facility location the client demands are expected to be correlated due to economic and/or geographic factors), which the independent-activation model clearly does not capture. A more general way of specifying the distribution is the *black-box model*, where the distribution is specified *only* via a procedure (a "black box") that one can use to independently sample scenarios from the distribution. In this model, each procedure call is treated as an elementary operation, and the running time of an algorithm is measured in terms of the number of procedure calls. The black-box model incorporates the desirable aspects of both the previous models: it allows one to specify distributions with exponentially many scenarios and correlation in a compact way that makes it reasonable to talk about polynomial-time algorithms.

Stochastic optimization problems are often computationally quite difficult, and often more difficult than their deterministic counterparts, both from the viewpoint of complexity theory, as well as from a practical perspective. In many settings the computational difficulty stems from the fact that the distribution might assign a non-zero probability to an exponential number of scenarios, leading to considerable increase in the problem complexity, a phenomenon often called the "curse of dimensionality." Thus, many stochastic problems that are easy to solve in the polynomial-scenario model due to the expansive input encoding become *NP*-hard in the black-box model. For example, *stochastic linear programming* problems (i.e., stochastic problems that can be formulated as linear programs) are polynomial-time solvable in the polynomial-scenario model but become #*P*-hard in the black-box model [8]. In other settings, even with polynomially many scenarios, the stochastic problem gives rise to a more complex problem than its deterministic counterpart and is *NP*-hard, whereas the deterministic problem is solvable in polynomial time.

In this survey, we focus on the design of approximation algorithms for stochastic optimization problems. Throughout, we use a *ρ-approximation algorithm* to denote a polynomial-time algorithm that always returns a feasible solution with objective function value within a factor ρ of the optimum; ρ is called the approximation ratio or performance guarantee of the algorithm.

There is an abundance of literature in the stochastic programming community that deals with computational aspects of solving 2-stage stochastic programs, especially 2-stage linear programs (LPs), which we shall not cover here; the reader is referred to [2,22] for more information. Many of these methods are only suitable in the polynomial-scenario model and cannot handle the burden of an exponential number of scenarios. One appealing approach in the black-box model is to sample a certain number of times from the scenario-distribution, estimate the probability of a scenario by its frequency in the sampled set, and solve the 2-stage problem determined by this approximate distribution. This is known as the *sample average approximation* (SAA) method. The SAA method is a widely used heuristic in practice and has been empirically shown to work well in various settings (see, e.g., [15,28]). The main question here is: how many samples does one need to ensure that an optimal solution to the *sample-average problem* is a near-optimal solution to the original problem (with high probability)? While there are results that prove asymptotic convergence to the optimal solution (to the original problem) in the limit *as the number of samples goes to infinity*, fewer results are known about the rate of convergence, or equivalently, about *worst-case bounds* on the sample size required to obtain a near-optimal solution. Ideally one would like to show that a polynomial number of samples always suffice. Such a result would show that the SAA method gives a reduction from the black-box problem to a polynomial-scenario problem, thereby reducing the complexity of the stochastic problem, while losing a factor in the approximation guarantee. In particular, this would immediately give an approximation algorithm for stochastic linear programming problems in the black-box model. The work that most closely considers the aspect of worst-case bounds is a paper of Kleywegt, Shapiro and Homem-De-Mello [14] (see also [23]). Kleywegt et al. prove a sample-size bound for 2-stage programs that is independent of the number of scenarios, but depends on the variance of a certain quantity (calculated using the scenario-distribution) which need not be polynomially bounded, even for very structured programs. We shall return to this question of proving polynomial sample-size bounds for the SAA method in Section 4.

There are other sampling-based approaches where instead of sampling just once initially, the algorithm used to solve the stochastic problem contains a sampling subroutine that is called whenever one needs to estimate some quantity, such as the function value or the gradient. Dyer, Kannan and Stougie [7] use such an approach for a stochastic maximization LP, where samples are used to estimate the objective function value at a given point. This yields a sample size that is only polynomial in the maximum value attained by any scenario (due to the high variance in the values of the different scenarios). Nesterov and Vial [20] employ stochastic subgradients, estimated via sampling, in a subgradient-descent algorithm, and require a sample size that is polynomial in the maximum variation in the objective function value in the feasible region.

The design and analysis of algorithms with provable worst-case guarantees for 2-stage stochastic integer programs is a relatively recent research direction. The first such result appears to be due to Dye, Stougie and Tomasgard [6] who give a constant-factor approximation algorithm for a resource provisioning problem in the polynomial-scenario model. Subsequently, a series of papers [21,11,10,25] appeared on this topic in the Computer Science literature, and showed that one can obtain guarantees for a variety of stochastic combinatorial optimization problems by adapting the techniques developed for the deterministic analogue. Gupta, Pál, Ravi and Sinha [10], who were the first to consider the black-box model (under a certain cost assumption), make such a connection explicit. Shmoys and

Swamy [25], who give algorithms in the black-box model with arbitrary costs, show an even more explicit correspondence. They showed that one could derive approximation algorithms for most of the problems considered in [21,11,10] by adopting a natural LP rounding approach that, in effect, converted an LP-based approximation guarantee for the deterministic analogue into a guarantee for the stochastic generalization with a small loss in the approximation factor. Thus, if we can solve the stochastic LP (even approximately), which is a $\#P$-hard problem, then we will have essentially reduced the stochastic problem to its deterministic analogue.

This survey is organized as follows: in Section 2 we describe an approximation scheme for solving a large class of 2-stage stochastic LPs. In Section 3 we describe some techniques for devising approximation algorithms for stochastic integer programming problems. We focus mainly on the black-box model, but also sometimes consider the polynomial-scenario model; in Section 4 we consider the SAA method and establish a concrete connection between these two models.

2 Stochastic Linear Programs

We now describe the fully polynomial approximation scheme (FPAS) of Shmoys and Swamy [25] that can be applied to a rich class of 2-stage stochastic LPs. The algorithm returns a solution of value within $(1+\kappa)$ times the optimum (with high probability), for any $\kappa > 0$, in time polynomial in the input size, $\frac{1}{\kappa}$, and a parameter λ, which is the maximum *ratio* between the second- and first-stage costs. As we show in Section 3, this provides us with a powerful and versatile tool for designing approximation algorithms for stochastic integer optimization problems in much the same way that linear programming has proved to be immensely useful in the design of approximation algorithms for deterministic optimization problems.

We shall consider a stochastic generalization of the set cover problem as an illustrative problem to explain the main ideas. In the *2-stage stochastic set cover* (SSC) problem, we are given a ground set U of n elements, a collection of subsets of U, $S_1, \ldots, S_m$, and a distribution over subsets of U that specifies the target set of elements to cover. In stage I, we can pick some sets paying a cost of w_S^{I} for each set S. Then, a scenario materializes which specifies a target set $A \subseteq U$ of elements to be covered and the costs $\{w_S^A\}$ of picking sets in that scenario, and one can pick additional sets to ensure that A is contained in the union of the sets selected in the two stages. The aim is to minimize the expected cost of the sets picked. Denoting the probability of scenario A by p_A (which we do not know explicitly, and could be 0), we can formulate the problem as an integer program and relax the integrality constraints to obtain the following linear program: minimize

$$\Big\{\sum_S w_S^{\mathrm{I}} x_S + \sum_{A \subseteq U, S} p_A w_S^A r_{A,S} : \sum_{S: e \in S} (x_S + r_{A,S}) \geq 1 \quad \forall A, e \in A; \quad x_S, r_{A,S} \geq 0 \quad \forall A, S. \Big\} \tag{SSC-P1}$$

Variable x_S indicates whether set S is chosen in stage I, and $r_{A,S}$ indicates if set S is chosen in scenario A. The constraint says that in every scenario A, every element in that scenario has to be covered by a set chosen either in stage I or in stage II. Notice that (SSC-P1) is an LP with an exponential number of variables *and* constraints, and it seems difficult to efficiently compute an (near-) optimal solution to (SSC-P1), since even writing out a solution can take exponential space (and time). However, if we fix the first-stage decisions, i.e., the x_S variables, then the scenarios become separable, so we can reformulate (SSC-P1) as follows: minimize

$$h(x) \;:=\; \sum_S w^{\mathrm{I}}_S x_S + \sum_{A\subseteq U} p_A f_A(x) \quad \text{subject to} \quad 0 \le x_S \le 1 \quad \forall S, \qquad \text{(SSC-P2)}$$

$$\text{where} \qquad (1)$$

$$f_A(x) \;:=\; \min \Bigl\{ \sum_S w^A_S r_{A,S} : \sum_{S:e\in S} r_{A,S} \ge 1 - \sum_{S:e\in S} x_S \; \forall e \in A; \quad r_{A,S} \ge 0 \; \forall S. \Bigr\}$$

Here the second-stage decisions only appear in the minimization problem $f_A(x)$, which denotes the recourse problem that one needs to solve for scenario A. It is easy to show that (SSC-P2) is equivalent to (SSC-P1), and that its objective function is convex. Although we now have a compact *convex program*, the complexity of the problem resurfaces as follows: in general, it is $\#P$-hard to even evaluate the objective function $h(.)$ at a given point [8]. Nevertheless, we can leverage convexity and adapt the *ellipsoid method* to solve (SSC-P2).

In the ellipsoid method, we start by containing the feasible region within a ball and then generate a sequence of ellipsoids, each of successively smaller volume. In each iteration, one examines the center of the current ellipsoid and obtains a specific half-space defined by a hyperplane passing through the current ellipsoid center. If the current ellipsoid center is infeasible, then one uses a violated inequality as the hyperplane, otherwise, one uses an *objective function cut* to eliminate (some or all) feasible points whose objective function value is no better than the current center, and thus make progress. A new ellipsoid is then generated by finding the minimum-volume ellipsoid containing the half-ellipsoid obtained by the intersection of the current one with this half-space. Continuing in this way, using the fact that the volume of the successive ellipsoids decreases by a significant factor, one can show that after a polynomial number of iterations, the feasible point generated with the best objective function value is a near-optimal solution.

Let $\mathcal{P} = \mathcal{P}_0$ denote the polytope $\bigl\{x \in \mathbb{R}^m : 0 \le x_S \le 1 \text{ for all } S\bigr\}$, and x_i be the current iterate. Define $\lambda = \max(1, \max_{A,S} w^A_S / w^{\mathrm{I}}_S)$, which we assume is known. It is trivial to determine if x_i is feasible, so we only need to describe how to obtain an objective function cut. One option is to simply add the constraint $h(x) \le h(x_i)$, which is not a "linear" cut, but would preserve the convexity of the feasible region. But then in subsequent iterations, without the ability to evaluate (or even estimate) $h(.)$ at a given point, we would not even be able to decide if the current point is feasible (or even almost-feasible), which poses a formidable difficulty. Alternatively, one could use cuts generated by a *subgradient*, which is the analogue of gradient for a non-differentiable function: $d \in \mathbb{R}^m$ is a subgradient of a function $g : \mathbb{R}^m \mapsto \mathbb{R}$ at point u, if $g(v) - g(u) \geq d \cdot (v - u)$ for every $v \in \mathbb{R}^m$. If d_i is a subgradient at point x_i, one can add the *subgradient cut* $d_i \cdot (x - x_i) \leq 0$ and proceed with the (smaller) polytope $\mathcal{P}_{i+1} = \{x \in \mathcal{P}_i : d_i \cdot (x - x_i) \leq 0\}$. Unfortunately, even computing a subgradient is hard to do in polynomial time for the objective functions that arise in stochastic programs. We circumvent this obstacle by using an *approximate subgradient*:

Definition 1. *We say that $\hat{d} \in \mathbb{R}^m$ is an $(\omega, \mathcal{D})$-subgradient of a function $g : \mathbb{R}^m \mapsto \mathbb{R}$ at point $u \in \mathcal{D}$, if for every $v \in \mathcal{D}$, we have $g(v) - g(u) \geq \hat{d} \cdot (v - u) - \omega g(u)$.*

We abbreviate $(\omega, \mathcal{P})$-subgradient to ω-subgradient. An extremely useful property of ω-subgradients is that one can compute them efficiently by sampling. If $\hat{d}_i$ is an ω-subgradient at x_i, one can add the inequality $\hat{d}_i \cdot (x - x_i) \leq 0$ and obtain the polytope $\mathcal{P}_{i+1} = \{x \in \mathcal{P}_i : \hat{d}_i \cdot (x - x_i) \leq 0\}$. Since we use an approximate subgradient, this might discard points with $h(.)$ value less than $h(x_i)$. But for any point $y \in \mathcal{P}_i \setminus \mathcal{P}_{i+1}$, we have that $h(y) \geq (1 - \omega)h(x_i)$, so no such point has $h(.)$ value much smaller than $h(x_i)$. Continuing this way, we obtain a polynomial number of points $x_0, x_1, \ldots, x_k$ such that $x_i \in \mathcal{P}_i \subseteq \mathcal{P}_{i-1}$ for each i, and the volume of the ellipsoid centered at x_k containing $\mathcal{P}_k$, and hence that of $\mathcal{P}_k$ is small. Now if $h(.)$ has a bounded Lipschitz constant (h has Lipschitz constant at most K if $|h(v) - h(u)| \leq \|v - u\|_2 \ \forall u, v \in \mathbb{R}^m$) then one can show that $\min_i h(x_i)$ is close to the optimal value OPT with high probability. The entire procedure is summarized below.

FindOpt(γ, ϵ) [Returns $\bar{x} \in \mathcal{P}$ such that $h(\bar{x}) \leq OPT/(1 - \gamma) + \epsilon$. Assume $\gamma \leq \frac{1}{2}$. K is the Lipschitz constant.]

```
O1. Set $k \leftarrow 0$, $y_0 \leftarrow \mathbf{0}$, $N \leftarrow \lceil 2m^2 \ln(\frac{16KR^2}{V\epsilon}) \rceil$, $n \leftarrow N \log(\frac{8NKR}{\epsilon})$, and $\omega \leftarrow \gamma/2n$.
    Let $E_0 \leftarrow B(\mathbf{0}, R)$ and $\mathcal{P}_0 \leftarrow \mathcal{P}$.
O2. For $i = 0, \ldots, N$ do the following.
    a) If $y_i \in \mathcal{P}_k$, set $x_k \leftarrow y_i$. Let $\hat{d}_k$ be an $\omega$-subgradient of $h(.)$ at $x_k$. Let $H$ denote
       the half space $\{x \in \mathbb{R}^m : \hat{d}_k \cdot (x - x_k) \leq 0\}$. Set $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cap H$ and $k \leftarrow k + 1$.
    b) If $y_i \notin \mathcal{P}_k$, let $a \cdot x \leq b$ be a violated inequality, that is, $a \cdot y_i > b$, whereas
       $a \cdot x \leq b$ for all $x \in \mathcal{P}_k$. Let $H$ be the half space $\{x \in \mathbb{R}^m : a \cdot (x - y_i) \leq 0\}$.
    c) Set $E_{i+1}$ to be the ellipsoid of minimum volume containing the half-ellipsoid
       $E_i \cap H$.
O3. Set $k \leftarrow k - 1$. Return the point in $\{x_0, \ldots, x_k\}$ with minimum $h(.)$ value.
```

There are a few details needed to complete the algorithm description. First, since we cannot compute $h(x)$ we will not be able to compute the point $\arg\min_i h(x_i)$ in step O3. Instead, by using ω-subgradients we will find a point $\bar{x}$ in the convex hull of $x_0, \ldots, x_k$, such that $h(\bar{x})$ is close to $\min_i h(x_i)$. We repeatedly perform a bisection search on the line segment joining $\bar{x}$ (initialized to x_0) and x_i for $i = 1, \ldots, k$, using an ω-subgradient to infer which direction to move along the segment. Each time the search returns a point y such that $h(y)$ is close to $\min(h(\bar{x}), h(x_i))$, and we update $\bar{x}$ to y. Second, to convert the performance guarantee of procedure $\mathsf{FindOpt}$ into a purely multiplicative $(1+\kappa)$-guarantee, we need to obtain a lower bound on OPT (and set ϵ, γ accordingly). Under the mild assumption that the cost of every set S, in stage I and in every stage II scenario, is at least 1, one can do this by sampling initially $O(\lambda)$ times. Essentially, one can detect by sampling $O(\lambda)$ times, whether the probability that some scenario $A \neq \emptyset$ occurs is at least $\frac{1}{\lambda}$; if so, then $OPT \geq \frac{1}{\lambda}$, otherwise $x = \mathbf{0}$ is an optimal solution. Finally, we specify how to compute an ω-subgradient at a point $x \in \mathcal{P}$ efficiently. Let z_A^* be an optimal solution to the *dual* of $f_A(x)$.

Lemma 1. *(i) the vector d with components $d_S = \sum_A p_A(w_S^{\mathrm{I}} - \sum_{e \in A \cap S} z_{A,e}^*)$ is a subgradient of $h(.)$ at x; (ii) for every scenario A and set S, $|w_S^{\mathrm{I}} - \sum_{e \in A \cap S} z_{A,e}^*| \leq \lambda w_S^{\mathrm{I}}$; and (iii) if $\hat{d} \in \mathbb{R}^m$ is such that $d_S - \omega w_S^{\mathrm{I}} \leq \hat{d}_S \leq d_S$ for every S, then $\hat{d}$ is an ω-subgradient of $h(.)$ at x.*

Parts (i) and (ii) of Lemma 1 show that each component of the subgradient vector is the expectation of a random variable (according to the scenario-distribution) with bounded variance. (Part (ii) also yields a bound on the Lipschitz constant of h.) So, with probability at least $1 - \delta$, one can estimate this expectation to within an additive error of ωw_S^{I} simultaneously for each S, using sample size $\mathsf{poly}\big(\text{input size}, \frac{\lambda}{\omega}, \ln(\frac{1}{\delta})\big)$. This yields an ω-subgradient, by part (iii) of Lemma 1. We compute an ω-subgradient at a polynomial number of points, with a polynomially small ω, so overall we get a sample size that is polynomial in the input size, λ, and $\frac{1}{\kappa}$. This sample-size bound is tight up to polynomial factors in the black-box model: one can construct examples where $\Omega(\lambda/\rho)$ samples are needed in the black-box model to obtain a performance guarantee of ρ [25], and the dependence on κ is also unavoidable due to the $\#P$-hardness result.

Shmoys and Swamy showed that the arguments above, especially Lemma 1, can be generalized to yield an approximation scheme for a broad class of 2-stage stochastic LPs which includes the fractional versions of a variety of stochastic combinatorial optimization problems such as (stochastic) covering problems (e.g., set cover, network design, multicut), facility location problems, multicommodity flow.

3 Stochastic Integer Programs

We now consider some stochastic combinatorial optimization problems, modeled as stochastic integer programs, and describe some methods that can be used to design approximation algorithms for these problems.

A general rounding technique. We first describe a simple, but powerful rounding framework due to [25], using stochastic set cover (SSC) as an illustrative example. Recall the relaxation (SSC-P2) for SSC. We will show that an LP-based approximation guarantee for the deterministic set cover (DSC) problem yields a corresponding guarantee for the stochastic problem. Given a DSC instance with a universe U of n elements, a family of subsets $S_1, \ldots, S_m$ with set S having weight w_S, consider the following LP relaxation of the integer problem of picking a minimum weight collection of sets to cover U.

$$OPT_{Det} \;:=\; \min \quad \sum_{S \in \mathcal{S}} w_S x_S \quad \text{subject to} \quad \sum_{S \in \mathcal{S}: e \in S} x_S \geq 1 \;\forall e; \quad x_S \geq 0 \;\forall S. \tag{SC-P}$$

Theorem 2. *Given an algorithm that for every DSC instance produces a solution of cost at most $\rho \cdot OPT_{Det}$, one can convert any solution x to* (SSC-P2) *to an integer solution of cost at most $2\rho \cdot h(x)$.*

Proof. Let r^*_A be an optimal solution to the recourse problem $f_A(x)$, so $f_A(x) = \sum_S w^A_S r^*_{A,S}$. Observe the following simple fact: an element e is covered to an extent of at least $\frac{1}{2}$ either by the variables x_S, or by the variables $r^*_{A,S}$ in *every scenario* A containing e. Let $E = \{e : \sum_{S: e \in S} x_S \geq \frac{1}{2}\}$. Then $(2x)$ is a fractional set cover solution for the instance with universe E, so one can obtain an integer set cover $\tilde{x}$ for E of cost at most $2\rho \cdot \sum_S w^{\mathrm{I}}_S x_S$. These are our stage I sets. Similarly, for any scenario A, $(2r^*_A)$ is a fractional set cover for $A \setminus E$, since for each such element e we have $\sum_{S: e \in S} r^*_{A,S} \geq \frac{1}{2}$. Therefore, one can cover these elements at a cost of at most $2\rho \cdot \sum_S w^A_S r^*_{A,S}$. So the cost of the solution $\tilde{x}$ is at most $2\rho \cdot h(x)$. □

Combined with the FPAS of Section 2, this yields approximation guarantees for various stochastic covering problems, e.g., we obtain guarantees of $2 \log n + \epsilon$ for SSC, and $4 + \epsilon$ for stochastic vertex cover.

Stochastic facility location. In the deterministic uncapacitated facility location (UFL) problem, given a set of candidate facilities $\mathcal{F}$ and a set of clients $\mathcal{D}$, we have to select a subset of facilities to open and assign each client to an open facility. Each facility i has an opening cost of f_i and each client j has demand d_j, and the cost of assigning client j to facility i is given by $d_j c_{ij}$, where c_{ij} is the distance between i and j and these distances form a metric. The goal is to minimize the sum of the facility opening and client assignment costs. In the 2-stage stochastic version of the problem, abbreviated SUFL, the demand of a client is a random variable (the demands may be correlated), and we can open facilities either in stage I, or after the scenario A with demands d^A_j is revealed, paying a cost of f^{I}_i or f^A_i respectively for opening facility i. We first consider SUFL in the polynomial-scenario model and show that one can design an approximation algorithm by dovetailing an approach used for UFL. Then we show that the above rounding technique can be adapted to derive an approximation algorithm for SUFL in the black-box model. For simplicity, we will assume that $d^A_j \in \{0, 1\}$ for every j, A, so a scenario now specifies a set of clients that need to be assigned to facilities.

Let $\mathcal{A}$ denote the collection of all scenarios, which is explicitly described in the input in the polynomial-scenario model. Consider the following LP relaxation for

SUFL. We use i to index the facilities, j to index the clients, and A to index the scenarios. Variables y_i and $y_{A,i}$ indicate whether facility i is opened in stage I or in scenario A respectively, and $x_{A,ij}$ indicates if client j is assigned to facility i in scenario A.

$$\begin{array}{lll}
\min \sum_i f_i^{\mathrm{I}} y_i + \sum_A p_A\Big(\sum_i f_i^A y_{A,i} + \sum_{j\in A,i} c_{ij}x_{A,ij}\Big) & & \text{(P)} \\
\text{s.t. } \sum_i x_{A,ij} \geq 1 & \forall A, j \in A & \\
x_{A,ij} \leq y_i + y_{A,i} & \forall i, A, j \in A & \\
y_i, x_{A,ij}, y_{A,i} \geq 0 & \forall i, A, j \in A. &
\end{array}$$

$$\begin{array}{llll}
\max & \sum_{A,j\in A} p_A\alpha_{A,j} & & \text{(D)} \\
\text{s.t.} & \alpha_{A,j} \leq c_{ij} + \beta_{A,ij} & \forall i, A, j \in A & (2) \\
& \sum_{j\in A} \beta_{A,ij} \leq f_i^A & \forall A, i & (3) \\
& \sum_{A,j\in A} p_A\beta_{A,ij} \leq f_i^{\mathrm{I}} & \forall i & (4) \\
& \alpha_{A,j}, \beta_{A,ij} \geq 0 & \forall i, A, j \in A. &
\end{array}$$

(D) is the dual program. We briefly sketch a primal-dual 3-approximation algorithm due to Mahdian [16], which closely resembles the Jain-Vazirani (JV) algorithm for UFL [12]. All dual variables are initially set to 0. It is easy to imagine the dual-ascent process: we uniformly increase all $\alpha_{A,j}$ variables at rate 1 until one of the constraints becomes tight. If constraint (2) goes tight for some (j, A) and facility i, we also start increasing $\beta_{A,ij}$ at rate 1. If constraint (3) goes tight for some A, i, then we tentatively open facility i for scenario A and freeze (i.e., stop increasing) all $\alpha_{A,j}, \beta_{A,ij}$ variables for which $\alpha_{A,j} \geq c_{ij}$. If (4) goes tight for a facility i, we tentatively open i for stage I, and for every scenario A, we freeze the $\alpha_{A,j}, \beta_{A,ij}$ variables for which $\alpha_{A,ij} \geq c_{ij}$. The process ends when all $\alpha_{A,j}$ variables are frozen. Now we perform a clean-up step for stage I, and for each scenario, to decide which facilities to open. For stage I, we open a maximal subset F of the tentatively open stage I facilities, such that for every (j, A), there is at most one facility $i \in F$ with $\beta_{A,ij} > 0$. In every scenario A, we open a maximal subset F_A of the tentatively open facilities for scenario A, such that for every $j \in A$, there is at most one facility $i \in F \cup F_A$ with $\beta_{A,ij} > 0$. The analysis proceeds as in the JV algorithm, by showing that for every (j, A), if the facility that caused $\alpha_{A,j}$ to freeze is not open, then there must be a facility opened in stage I or in scenario A that is at most $3\alpha_{A,j}$ distance away from j. This proves an approximation ratio of 3.

We now consider SUFL in the black-box model. We compactify (P) to obtain the convex program: minimize $h(y) := \sum_i f_i^{\mathrm{I}} y_i + \sum_{A\in\mathcal{A}} p_A g_A(y)$, where $g_A(y)$ is the minimum of $\sum_i f_i^A y_{A,i} + \sum_{j\in A,i} c_{ij}x_{A,ij}$ subject to the constraints $\sum_i x_{A,ij} \geq 1$ for all $j \in A$, $x_{A,ij} \leq y_i, y_{A,i}$ for all $i, j \in A$, and $x_{A,ij}, y_{A,i} \geq 0$ for all $i, j \in A$. Note that this is not a stochastic covering program. While UFL admits a star-covering relaxation (clients have to be covered by stars, a star is a facility and a set of clients assigned to it), the corresponding stochastic covering program does not model SUFL, because in SUFL when we open a facility in stage I we do not fix then the set of clients it will serve; this is decided in stage II, and will typically be scenario-dependent. Yet, the above rounding technique can be adapted here, by applying decoupling to the covering constraint

$\sum_i x_{A,ij} \geq 1$. Let ρ_{UFL} denote the integrality gap of UFL, which is at most 1.52 [17].

Theorem 3. *The integrality gap of (P) is at most* $2\rho_{\mathsf{UFL}}$.

Proof. Let y be any feasible solution to the convex program and let (x^*_A, y^*_A) be an optimal solution to $g_A(y)$. We write $x^*_{A,ij} = x^{\mathrm{I}}_{A,ij} + x^{\mathrm{II}}_{A,ij}$ for each scenario A and client $j \in A$, where $x^{\mathrm{I}}_{A,ij} \leq y^*_i$ and $x^{\mathrm{II}}_{A,ij} \leq y^*_{A,i}$. This is always possible since $x^*_{A,ij} \leq y^*_i + y^*_{A,i}$. So either $\sum_i x^{\mathrm{I}}_{A,ij} \geq \frac{1}{2}$ or $x^{\mathrm{II}}_{A,ij} \geq \frac{1}{2}$. For a client j, define $\mathcal{S}_j = \{A \ni j : \sum_i x^{\mathrm{I}}_{A,ij} \geq \frac{1}{2}\}$. For the stage I decisions, we construct a feasible fractional solution for a UFL instance where the facility costs are f^{I}_i, the assignment costs are c_{ij}, and the "demand" of client j is set to $\sum_{A \in \mathcal{S}_j} p_A$, and then round this using an algorithm for UFL. If we treat each (j, A) where $A \in \mathcal{S}_j$ as a separate client with demand p_A, we obtain a feasible solution by setting $\hat{y}_i = \min(1, 2y^*_i)$ and $\hat{x}_{A,ij} = \min(1, 2x^{\mathrm{I}}_{A,ij})$. yields a feasible solution for this instance. But since the $\hat{y}_i$ facility variables do not depend on the scenario, we can re-optimize the assignment for each (j, A) to obtain an assignment that does not depend on A. Thus, we can coalesce all the (j, A) clients into one, with demand $\sum_{A \in \mathcal{S}_j} p_A$. $2(\sum_i f^{\mathrm{I}}_i y^*_i + \sum_{j,i,A \in \mathcal{S}_j} p_A c_{ij} x^{\mathrm{I}}_{A,ij})$. Since the integrality gap is ρ_{UFL}, there is an integer solution $(\tilde{x}, \tilde{y})$ of cost at most $2\rho_{\mathsf{UFL}}(\sum_i f^{\mathrm{I}}_i y^*_i + \sum_{j,i,A \in \mathcal{S}_j} p_A c_{ij} x^{\mathrm{I}}_{A,ij})$; this determines which facilities to open in stage I. In any scenario A, each client j such that $A \in \mathcal{S}_j$ is assigned to the stage I facility given by the assignment $\tilde{x}$. For each remaining client j, since $\sum_i x^{\mathrm{II}}_{A,ij} \geq \frac{1}{2}$, the solution $\hat{y}_{A,i} = \min\bigl(1, 2y^*_{A,i}\bigr)$, $\hat{x}_{A,ij} = \min\bigl(1, 2x^{\mathrm{II}}_{A,ij}\bigr)$ yields a feasible solution for the UFL instance with client set $\{j \in A : A \notin \mathcal{S}_j\}$. Again the ρ_{UFL} integrality gap shows that there is an integer solution with "low" cost. Overall, we get that the total cost of the solution $\tilde{y}$ is at most $2\rho_{\mathsf{UFL}} \cdot h(y)$. This shows that the integrality gap is at most $2\rho_{\mathsf{UFL}}$ (and gives a 3.04-approximation algorithm in the polynomial-scenario model (taking $\rho_{\mathsf{UFL}} = 1.52$).) □

The rounding approach does not yet yield the strongest performance guarantee currently known. We will return to this problem in Section 4.

Stochastic Steiner tree. We now describe the boosted sampling technique of Gupta et al. [10] that shows that for certain stochastic problems, an approximation algorithm for the deterministic problem that satisfies some *cost-sharing* properties, can be used to derive performance guarantees for the stochastic problem. We focus on the stochastic rooted Steiner tree (SST) problem: we have a graph $G = (V, E)$, a fixed root $r \in V$, and a distribution that specifies a random set of terminals to connect to the root. We can buy edges either in stage I or after the terminal set $A \subseteq V$ has been revealed, paying a cost of c_e or c^A_e respectively for edge e, so as to connect all the nodes in A to r. It is worth noting that we can formulate a *fractional version* of SST as a stochastic covering problem, where each cut separating a terminal from the root must be covered by edges bought in the two stages. One can therefore obtain a $(1 + \epsilon)$-optimal fractional solution in polynomial time. However the rounding procedure detailed above does not work, because the cut-covering problems obtained after decoupling the two stages need not correspond to Steiner tree instances (and may not even fall into the Goemans-Williamson framework [9]). We can use boosted sampling to devise a 4-approximation algorithm, under the cost restriction $c_e^A = \lambda^A c_e$ for every edge e in every scenario A. This reflects a limitation of the boosted sampling approach. In the case of SST, without such a restriction the problem becomes Group-Steiner-tree-hard [21], but for other problems such as stochastic {vertex cover, facility location}, one can obtain good guarantees *without imposing any cost restrictions* by using other techniques.

Here we assume for simplicity that $c_e^A = \lambda c_e$ for every A, e. Let $\mathsf{ST}(S)$ denote the cost of an optimal Steiner tree on $S \cup \{r\}$ wrt. costs $\{c_e\}$. We say that an algorithm $\mathcal{A}$ for the Steiner tree problem admits a *β-strict cost sharing* if there is a function $\xi : 2^V \times V \mapsto \mathbb{R}_{\geq 0}$ such that for every $S, T \subseteq V$ with $S \cap T = \emptyset$, (i) $\xi(S, u) = 0$ for $u \notin S$; (ii) $\sum_{u \in S} \xi(S, u) \leq \mathsf{ST}(S)$; and (iii) there is a procedure $\mathsf{Aug}_{\mathcal{A}}$ that augments the tree $\mathcal{A}(S)$ constructed by $\mathcal{A}$ on input S to a tree on $S \cup T \cup \{r\}$ incurring cost $c\big(\mathsf{Aug}_{\mathcal{A}}(S, T)\big) \leq \beta \sum_{u \in T} \xi(S \cup T, u)$. Intuitively $\xi(S, u)$ stands for u's share in the cost of a Steiner tree on S.

We may assume that G is complete and the edge costs form a metric. We use the MST heuristic as algorithm $\mathcal{A}$. This is a 2-approximation algorithm that admits a 2-strict cost sharing. Procedure $\mathsf{Aug}_{\mathcal{A}}$ consists of contracting S into the root, and building an MST on $T \cup \{r\}$ in the contracted graph. Rooting the MST on $S \cup \{r\}$ at r, we set $\xi(S, u) = \frac{1}{2}$(cost of the edge joining u to its parent). This satisfies properties (i) and (ii) above, and it is not hard to show that it satisfies (iii) with $\beta = 2$. The algorithm for SST is quite simple and extremely elegant: we draw λ samples $A_1, \ldots, A_\lambda$ from the distribution and build the tree $\mathcal{A}(S)$ where $S = \bigcup_i A_i$, as our first-stage solution. Intuitively, this tries to account for the λ inflation factor by sampling each scenario A, in expectation, λp_A times. In the second-stage, if scenario A is realized, we use $\mathsf{Aug}_{\mathcal{A}}$ to augment $\mathcal{A}(S)$ and connect $A \setminus S$ to the root. A nice feature of the algorithm is that only λ samples are required.

Let E_1^* and E_A^* be the edges purchased in stage I and in scenario A by an optimal (integer) solution to SST, and let $OPT = c(E_1^*) + \lambda \mathrm{E}_A\big[c(E_A^*)\big]$ be the cost incurred. Let $\xi(X, Y)$ denote $\sum_{u \in Y} \xi(X, u)$. The first-stage cost can be bounded by noting that $\mathsf{ST}(S)$ is at most the cost of $Z_S = E_1^* \cup \big(\bigcup_{i=1}^{\lambda} E_{A_i}^*\big)$ since Z_S connects S to r. The expected first-stage cost is at most $2\mathrm{E}_S\big[\mathsf{ST}(S)\big] \leq 2\mathrm{E}_S\big[c(Z_S)\big]$ which is at most $2 \cdot OPT$, since each scenario A is sampled λp_A times in expectation. The expected second-stage cost is given by $\lambda \mathrm{E}_{S,A}\big[c(\mathsf{Aug}_{\mathcal{A}}(S, A \setminus S))\big]$ which is at most $2\lambda \mathrm{E}_{S,A}\big[\xi(S \cup A, A \setminus S)\big]$ by property (iii). We can treat scenario A as an extra sample $A_{\lambda+1}$, and since the A_i's are identically distributed, we have that $\mathrm{E}_{S,A}\big[\xi(S \cup A, A \setminus S)\big] \leq \frac{1}{\lambda+1}\mathrm{E}_{S,A}\big[\xi(S \cup A, S \cup A)\big] \leq \frac{1}{\lambda+1}\mathrm{E}_{S,A}\big[\mathsf{ST}(S \cup A)\big]$. Finally, by arguing as we did for stage I, one can bound $\mathrm{E}_{S,A}\big[\mathsf{ST}(S \cup A)\big]$ by $\frac{\lambda+1}{\lambda} \cdot OPT$. Thus the expected second-stage cost is at most $2 \cdot OPT$, and the total cost is at most $4 \cdot OPT$.

Gupta et al. showed that boosted sampling can be applied to any stochastic problem satisfying a certain sub-additivity condition, if we have an approximation algorithm for the deterministic version that admits a β-strict cost-sharing (which is now defined more abstractly). They show that an α-approximation algorithm with a β-strict cost sharing gives an $(\alpha + \beta)$-approximation algorithm for the stochastic problem. In all known cases, such an approximation algorithm is obtained via the primal-dual schema and the cost shares are derived from the dual variables. Thus, boosted sampling can be viewed as a primal-dual approach for designing approximation algorithms for stochastic problems.

4 The Sample Average Approximation Method

The sample average approximation (SAA) method is a natural approach for computing solutions in the black-box model. Here we replace the original stochastic problem by a sample-average problem obtained by sampling scenarios some $\mathcal{N}$ times and estimating the scenario probabilities by their frequencies of occurrence in the sampled set, and solve this problem. If one can show that a polynomially bounded $\mathcal{N}$ suffices to obtain a $(1+\epsilon)$-optimal solution (to the original problem), then one would obtain a reduction from the black-box problem to a polynomial-scenario problem while losing a $(1+\epsilon)$ factor. As mentioned earlier, Kleywegt et al. [14] prove a sample-size bound for general 2-stage programs that depends on the variance of a certain quantity, which need not be polynomially bounded. Although this bound is tight in the black-box model for general 2-stage programs [24], for structured programs such as the class of 2-stage LPs considered in [25], one can prove better bounds that do not follow directly from the bound in [14]. Swamy and Shmoys [27] gave a polynomial bound for this class by building upon ideas used in the ellipsoid-based FPAS of Section 2. More recently, Nemirovskii & Shapiro [19] showed that for the stochastic set cover problem, additional analytical insights yield similar bounds as a further consequence of the results of [14]. Thus, the SAA method yields a simpler, more efficient scheme for this class of programs.

The proof in [27] uses (approximate) subgradients to identify a notion of closeness between the sample-average and true objective functions. Loosely speaking, this notion captures the property that the ellipsoid-based FPAS can be made to run identically on both the sample-average and the true problems, which intuitively suggests that optimizing the sample-average function is nearly equivalent to optimizing the true function. Subsequently Charikar, Chekuri and Pál [3] gave a different proof for roughly the same class of programs. While [27] only shows that any *optimal* solution to the sample-average LP is a $(1+\epsilon)$-optimal solution to the true LP (with high probability), Charikar et al. argue that by slightly modifying the "standard" SAA approach, one can prove that any α-optimal solution to the sampled problem is an $(\alpha+\epsilon)$-optimal solution to the true problem. This implies the remarkable consequence that one can, in effect, reduce the black box model (for a class of 2-stage recourse minimization problems) to the polynomial-scenario model. In Section 3, we gave a 3-approximation algorithm for SUFL in the polynomial-scenario model. Likewise, by mimicking the primal-dual algorithm for vertex cover one can obtain the same guarantee of

2 for the stochastic problem in the polynomial-scenario model [21]. The above result shows that these guarantees also extend to the black-box model.

We have described a variety of techniques for the design of approximation algorithms for 2-stage stochastic linear and integer programs. This thread of algorithmic analysis of stochastic optimization approximation algorithms has the potential to bring together the insights and approaches from several disjoint research communities: (traditional) stochastic programming, theoretical computer science, and machine learning (where the flavor of learning a distribution based on a limited number of samples plays a central role). There is much more work remaining than has already been done, since the bulk of the work done thus far in such black box settings does not extend to a variable number of stages, or to settings beyond the simple expectation minimization objective.

References

1. E. M. L. Beale. On minimizing a convex function subject to linear inequalities. *J. Royal Stat. Soc., Series B*, 17:173–184; discussion 194–203, 1955.
2. J. R. Birge and F. V. Louveaux. *Introduction to Stochastic Programming.* Springer-Verlag, NY, 1997.
3. M. Charikar, C. Chekuri, and M. Pál. Sampling bounds for stochastic optimization. *Proc. 9th RANDOM*, 257–269, 2005.
4. G. B. Dantzig. Linear programming under uncertainty. *Management Sci.*, 1:197–206, 1955.
5. B. Dean, M. X. Goemans, and J. Vondrak. Approximating the stochastic knapsack problem: the benefit of adaptivity. *Proc. 45th Annual IEEE Symposium on Foundations of Computer Science*, 208–217, 2004.
6. S. Dye, L. Stougie, and A. Tomasgard. The stochastic single resource service-provision problem. *Naval Res. Logistics*, 50:869–887, 2003. Also appeared as "The stochastic single node service provision problem", COSOR-Memorandum 99-13, Dept. Math. & Comp. Sc., Eindhoven Tech. Univ., Eindhoven, 1999.
7. M. Dyer, R. Kannan, and L. Stougie. A simple randomised algorithm for convex optimisation. SPOR-Report 2002-05, Dept. Math. & Comp. Sc., Eindhoven Tech. Univ., Eindhoven, 2002.
8. M. Dyer and L. Stougie. Computational complexity of stochastic programming problems. SPOR-Report 2005-11, Dept. Math. & Comp. Sc., Eindhoven Tech. Univ., Eindhoven, 2005.
9. M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM J. Comput.*, 24:296–317, 1995.
10. A. Gupta, M. Pál, R. Ravi, and A. Sinha. Boosted sampling: approximation algorithms for stochastic optimization. *Proc. 36th ACM STOC*, 417–426, 2004.
11. N. Immorlica, D. Karger, M. Minkoff, and V. Mirrokni. On the costs and benefits of procrastination: approximation algorithms for stochastic combinatorial optimization problems. *Proc. 15th SODA*, 684–693, 2004.
12. K. Jain and V.V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and Lagrangian relaxation. *J. ACM*, 48:274–296, 2001.
13. J. Kleinberg, Y. Rabani, and É. Tardos. Allocating bandwidth for bursty connections. *SIAM J. Comput.*, 30:191–217, 2000.

14. A. J. Kleywegt, A. Shapiro, and T. Homem-De-Mello. The sample average approximation method for stochastic discrete optimization. *SIAM J. Optimization*, 12:479–502, 2001.
15. J. Linderoth, A. Shapiro, and S. Wright. The empirical behavior of sampling methods for stochastic programming. *Annals Oper. Res.*, to appear.
16. M. Mahdian. *Facility Location and the Analysis of Algorithms through Factor-revealing Programs.* Ph.D. thesis, MIT, Cambridge, MA, 2004.
17. M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location. *Proc. 5th APPROX*, 229–242, 2002.
18. R. Möhring, A. Schulz, and M. Uetz. Approximation in stochastic scheduling: the power of LP based priority policies. *JACM*, 46:924–942, 1999.
19. A. Nemirovski and A. Shapiro. On complexity of Shmoys–Swamy class of two-stage linear stochastic programming problems. *Optimization Online,* 2006. http://www.optimization-online.org/DB_FILE/2006/07/1434.pdf.
20. Y. Nesterov and J.-Ph. Vial. Confidence level solutions for stochastic programming. CORE Discussion Papers, 2000. http://www.core.ucl.ac.be/services/psfiles/dp00/dp2000-13.pdf.
21. R. Ravi and A. Sinha. Hedging uncertainty: approximation algorithms for stochastic optimization problems. *Proceedings, 10th IPCO*, 101–115, 2004.
22. A. Ruszczynski and A. Shapiro. Editors, *Stochastic Programming*, Volume 10 of *Handbooks in Oper. Res. & Mgmt. Sc.*, North-Holland, Amsterdam, 2003.
23. A. Shapiro. Monte Carlo sampling methods. In A. Ruszczynski and A. Shapiro, editors, *Stochastic Programming*, volume 10 of *Handbooks in Oper. Res. & Mgmt. Sc.*, North-Holland, Amsterdam, 2003.
24. A. Shapiro and A. Nemirovski. On complexity of stochastic programming problems. *Optimization Online*, 2004. http://www.optimization-online.org/DB_FILE/2004/10/978.pdf.
25. D. B. Shmoys and C. Swamy. An approximation scheme for stochastic linear programming and its application to stochastic integer programs. *J. ACM*, to appear. Preliminary version appeared as “Stochastic optimization is (almost) as easy as deterministic optimization” in *Proc. 45th Annual IEEE FOCS*, 228–237, 2004.
26. C. Swamy. *Approximation Algorithms for Clustering Problems.* Ph.D. thesis, Cornell Univ., Ithaca, NY, May 2004. http://www.math.uwaterloo.ca/~cswamy/theses/master.pdf.
27. C. Swamy and D. B. Shmoys. The sample average approximation method for 2-stage stochastic optimization. November 2004. http://www.math.uwaterloo.ca/~cswamy/papers/SAAproof.pdf.
28. B. Verweij, S. Ahmed, A. J. Kleywegt, G. L. Nemhauser, and A. Shapiro. The sample average approximation method applied to stochastic routing problems: a computational study. *Comp. Opt. Appl.*, 24:289–333, 2003.

The Number of Crossing Free Configurations on Finite Point Sets in the Plane

Emo Welzl

Institute of Theoretical Computer Science
ETH Zurich, Switzerland

Abstract. We consider the family of crossing-free geometric graphs of a certain type—most notably triangulations, but also spanning (Hamiltonian) cycles, spanning trees, matchings, etc.—on a given point set in the plane. In particular, we address the question of how large these families can be in terms of the number of points. After the issue was raised for Hamiltonian cycles by Newborn and Moser, and for triangulations by Avis, it was shown in 1982 by Ajtai, Chvátal, Newborn, and Szemerédi that for any set of n points the number of *all* crossing-free geometric graphs on is at most c^n for $c = 10^{13}$ (as opposed to the previously known bounds of the form $c^{n \log n}$). We report on some of the developments since then, e.g. a 43^n bound on the number of triangulations whose proof takes a detour via random triangulations.

While this problem seems elusive despite of some progress, related algorithmic questions are even less understood: For example the complexity of determining or approximating the number of triangulations of a point set, or generating a triangulation uniformly at random from all triangulations of a point set.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, p. 20, 2006.

Normal and Feature Approximations from Noisy Point Clouds[*]

Tamal K. Dey and Jian Sun

The Ohio State University, Columbus OH 43210, USA
{tamaldey, sunjia}@cse.ohio-state.edu

Abstract. We consider the problem of approximating normal and feature sizes of a surface from point cloud data that may be noisy. These problems are central to many applications dealing with point cloud data. In the noise-free case, the normals and feature sizes can be approximated by the centers of a set of unique large Delaunay balls called *polar* balls. In presence of noise, polar balls do not necessarily remain large and hence their centers may not be good for normal and feature size approximations. Earlier works suggest that some large Delaunay balls can play the role of polar balls. However, these results were short in explaining how the big Delaunay balls should be chosen for reliable approximations and how the approximation error depends on various factors. We provide new analyses that fill these gaps. In particular, they lead to new algorithms for practical and reliable normal and feature approximations.

1 Introduction

Recently, a number of algorithms have been designed for processing point cloud data. Often these algorithms, as a basic step, estimate the normals and features of the sampled surface from the given point cloud. For example, some algorithms [1,8,11] need a normal estimation step for surface reconstruction, and others estimate the scale of local geometry also called the local feature size to handle non-uniform samples [9,14]. In the noise-free case the problem of normal and feature size approximations have been well studied [2,4,6]. In the case of noise, optimization based techniques [1,13] are known for normal approximations though they do not have theoretical guarantees. It is known that results from the noise-free case can be extended by using big Delaunay balls that can help in estimating normals [8] with theoretical guarantees. However, it is not known how the error of approximation depends on different noise components, and more importantly, how the big Delaunay balls should be chosen for reliable approximations. The problem for feature approximations in presence of noise is much less understood. No reliable and practical algorithm is known for it. In this paper we address these open issues.

[*] This work is partially supported by NSF CARGO grant DMS-0310642 and ARO grant DAAD19-02-1-0347.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 21–32, 2006.

Motivation and results. Amenta and Bern [2] introduced the concept of poles. These are the furthest Voronoi vertices from the respective sites on the two sides of the sampled surface. In terms of the Delaunay triangulation, poles are the centers of the largest Delaunay balls incident to the sample points on both sides of the sampled surface. These balls are also called the polar balls. Amenta and Bern showed that, in the noise-free case, the normals can be estimated by the poles. Further, Amenta, Choi, Kolluri [4] and Boissonnat, Cazals [6] proved that the poles also approximate the medial axis and hence local feature sizes can be estimated by distances to the poles.

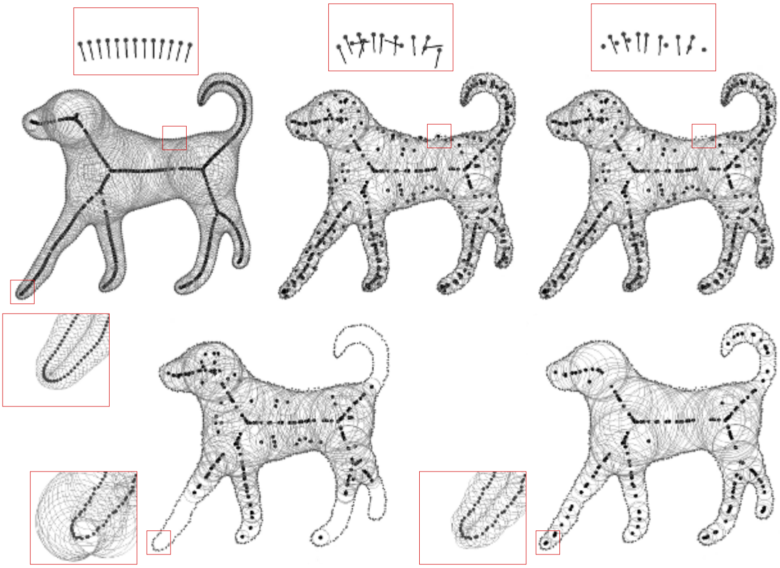


Fig. 1. Top row: Left: noise-free case, poles are approximating the medial axis and normals well. Middle: A small noise disturbs the poles significantly resulting in poor normal and medial axis approximation with all poles. Right: only a subset of big Delaunay balls are chosen, normals though not medial axis are well approximated. Bottom row: Left: Delaunay balls of bigger size are chosen to exclude unwanted poles, some significant parts of the medial axis are not approximated. Right: Centers of polar balls chosen with our algorithm approximate the medial axis everywhere. Approximated feature sizes are indicated in the highlighted boxes.

In the presence of noise, the above results do not hold since some of the polar balls can be arbitrarily small with their centers being arbitrarily close to the surface. See the top-middle picture in Figure 1. Nevertheless, Dey and Goswami [8] observed that, under a reasonable noise model, many Delaunay balls remain big and their centers can help in approximating the normals. The error in the normal approximation by big Delaunay balls obviously depends on the sampling density (ε) and also on the size of the chosen Delaunay balls. A detailed analysis on these dependencies is missing in earlier works. First, our analysis provides an error bound that unifies earlier results. Second, it tells us how to choose big Delaunay balls in practice for reliable normal and, in particular, feature size approximations for noisy point clouds.

In the noise-free case the choice of the Delaunay balls is not an issue in normal and feature size approximations as they are approximated from polar balls which are almost as big as medial balls. However, in the case of noise, the choice of Delaunay balls is an issue as all polar balls are not big. To remain scale independent one can choose Delaunay balls whose radii are larger than a threshold determined by some nearest neighbor distances of the sample points incident on the Delaunay balls. In order to gauge the viable values of the threshold, it is important to know how the normal and feature size approximation errors depend on the radii of the Delaunay balls. Our new analysis provides this relation. We show that normals can be estimated from Delaunay balls that are not necessarily as big as local feature sizes ($f(\cdot)$). In fact, Delaunay balls with radii as small as $\varepsilon^{\frac{1}{2}} f(\cdot)$ are also good for normal estimations. See top row in Figure 1 for an illustration.

The case for feature estimations in presence of noise is far more difficult. This is because, unlike normal approximations, not all centers of Delaunay balls chosen with a reasonable threshold approximate the medial axis. Choosing the right ones is hard. If the threshold is relatively small, a number of centers remain which do not approximate the medial axis. See top-right picture in Figure 1. On the other hand if the threshold is large, the medial axis for some parts of the models may not be approximated at all; see bottom-left picture in Figure 1. As a result no threshold may exist for which large Delaunay balls' centers approximate the medial axis, the Dog data in Figure 1 and the Horse data in Figure 4 are two such examples in two and three dimensions respectively.

We propose a different algorithm to choose the Delaunay balls for approximating the medial axis. We consider k-nearest neighbors for some k and take the largest polar ball's center among these neighbors to approximate the medial axis. Our analysis leads to this algorithm which frees us from the burden of choosing a size threshold. Our experiments suggest that k can be chosen fairly easily, generally in the range of 5 to 10. The most important thing is that a k can be found for which the medial axis is well approximated where no such size threshold may exist. The bottom row of Figure 1 illustrates this point.

Previous results. Amenta, Bern and Eppstein [3] introduced the ε-sampling for noise-free case. This requires each point on the surface to have a sample point within a distance of ε times the local feature size. When noise is allowed, the sample points need not lie exactly on the surface and may scatter around it. Therefore, the sampling model needs to specify both a *tangential scatter*, i.e., the sparseness of the sampling along tangent directions of the surface and also a *normal scatter*, i.e., the sparseness of sampling along the normal directions. Dey and Goswami [8] introduced a noise model that uses the same sampling parameter ε for both scatters. Kolluri [11] and later Dey and Sun [9] modified the normal scatter to have ε^2 dependence. The errors of normal and feature approximations depend on both tangential and normal scatters. Therefore, we introduce two independent parameters ε and δ for these two scatters to reveal the dependence of the approximation errors on these two parameters separately.

Normal approximation: Dey and Goswami [8] and Mederos et al. [12] showed that when both tangential and normal scatters are $O(\varepsilon)$ times the local feature size, the normals can be approximated with an $O(\sqrt{\varepsilon})$ error if the chosen Delaunay balls have radius almost as big as the local feature size. Dey and Sun [9] showed that the error is $O(\varepsilon)$ if the normal scatter is only $O(\varepsilon^2)$ times the local feature size. None of these results specify how the error depends on the radii of the chosen Delaunay balls.

In this paper we provide a simple elegant analysis which shows that the error is $2(\frac{1}{\lambda}+1)O(\varepsilon+\sqrt{\delta})$ where λ is the radius of the Delaunay ball. Previous results under different noise models can be derived from this unified result. One implication of this result is that Delaunay balls as small as $O(\varepsilon^{\frac{1}{2}}+\delta^{\frac{1}{4}})f(\cdot)$ can help in estimating the normals. This relaxes the burden on setting the parameter for the normal estimation algorithm.

Feature approximation: Amenta, Bern and Eppstein [3] defined the local feature size of a point x on the surface as the distance of x to the medial axis. Obviously, the local feature size can be estimated if the medial axis can be approximated. An algorithm for approximating the medial axis from noisy point clouds exists [7]. This algorithm approximates the medial axis with Voronoi faces under a stringent uniform sampling condition. Selecting Voronoi faces to approximate the medial axis is not a simple task in practice even for noise-free case [5,10] and it is not clear how this algorithm works in practice when noise is present. Moreover, for estimating the local feature size a continuous approximation with Voronoi faces is an overkill. A discrete approximation of the medial axis with a set of Voronoi vertices serves the purpose equally well. For the noise-free case, such an approximation was proposed by Amenta et al. [4] and Boissonnat and Cazals [6]. Recently, Mederos et al. [12] derived some results for noisy point clouds that have some connections to the local feature size approximations though the approximation factor depends on a surface related constant which can be potentially huge.

Our analysis is free of any surface dependent constant and it relates the approximation error to the tangential and normal scatters separately. Most importantly, the analysis justifies our choice of polar balls based on nearest neighbors to approximate the medial axis. Figure 1 and 4 show that this choice is far more superior than the big Delaunay ball strategy. Experiments with our implementation [16] of the algorithm confirm this claim for other models. Due to the space limitation, the proofs of our results cannot be included in this paper. An extended version including all the proofs is available from authors' webpages [15].

2 Preliminaries

2.1 Definitions

For a set $Y \subseteq \mathbb{R}^3$ and a point $x \in \mathbb{R}^3$, let $d(x, Y)$ denote the Euclidean distance of x from Y; that is,

$$d(x, Y) = \inf_{y \in Y} \{\|y - x\|\}.$$

The set $B_{c,r} = \{y \,|\, y \in \mathbb{R}^3, \|y - c\| \leq r\}$ is a *ball* with radius r and center c.

Voronoi and Delaunay diagram. For a finite point set $P \subset \mathbb{R}^3$, we will denote the Voronoi diagram and its dual Delaunay triangulation of P by $\operatorname{Vor} P$ and $\operatorname{Del} P$ respectively. The Voronoi cell for a point p is denoted as V_p.

Sampled surface. Let $\Sigma \subset \mathbb{R}^3$ be a compact smooth surface without boundary from which the input sample is derived possibly with noise. Also, assume that Σ is connected. The bounded and unbounded components of $\mathbb{R}^3 \setminus \Sigma$ are denoted Ω_I and Ω_O respectively. The normal at any point $x \in \Sigma$ is denoted $\mathbf{n}_x$ which is directed locally inward, i.e., toward Ω_I.

The *medial axis* M of Σ is the locus of the centers of the maximal balls whose interiors are empty of points in Σ. These balls meet Σ only tangentially. We call each such ball $B_{m,r}$ a *medial ball* where $r = d(m, \Sigma)$. Barring some pathological cases, we can assume $M \cap \Sigma$ is empty if Σ is smooth. The subsets of M in Ω_I and Ω_O are called *inner* and *outer* medial axis respectively. For each point $x \in \Sigma$, there are two *medial balls*, one centering a point in the inner medial axis and the other in the outer medial axis. The *local feature size* at a point $x \in \Sigma$ is defined as $f(x) = d(x, M)$. The function $f()$ satisfies the following Lipschitz property [2].

Lipschitz property. For any two points $x, y \in \Sigma$, $f(x) \leq f(y) + \|x - y\|$.

2.2 Sampling

A finite set of points $P \subset \Sigma$ is called an ε-sample of Σ if $d(x, P) \leq \varepsilon d(x, M)$ for each $x \in \Sigma$. To accommodate the tangential and normal scatters of points around Σ in the noisy case, we put two conditions on the sampling. The first condition says that the projection of the point set P on the surface makes a dense sample and the second one says that P is close to the surface. We also use a third condition to make the sampling locally uniform. To make the sampling definition general, we use a separate parameter for each sampling condition. For any point $x \in \mathbb{R}^3 \setminus M$ let $\tilde{x}$ denote its closest point on Σ. Clearly, the segment $x\tilde{x}$ is parallel to the normal $\mathbf{n}_{\tilde{x}}$.

We say $P \subset \mathbb{R}^3$ is a $(\varepsilon, \delta, \kappa)$-sample of Σ if the following conditions hold.

(i) $\tilde{P} = \{\tilde{p}\}_{p \in P}$ is an ε-sample of Σ,
(ii) $\|p - \tilde{p}\| \leq \delta f(\tilde{p})$,
(iii) $\|p - q\| \geq \varepsilon f(\tilde{p})$ for any two points p, q in P where q is the κth nearest sample point to p.

Figure 2 illustrates why we put the third condition. In the figure the same point sample satisfies the first two conditions for two different curves; C and also $C \cup C'$. Our analyses for normal and medial axis approximations apply to both of these curves; albeit with different scales of local feature sizes. Therefore, the analyses do not need the third condition in the sampling. However, our approximation algorithms determine a particular scale by looking at the nearest neighbor distances. This implies that the sampling cannot allow the ambiguity which is forced by assuming a local uniformity constraint in the third one.

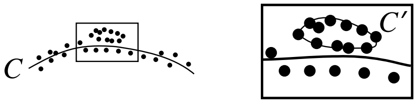


Fig. 2. A point sample satisfying sampling conditions (i) and (ii) for a single component curve C (left) and also the curve $C \cup C'$ (right)

In the analysis we concentrate only in the bounded component Ω_I together with the inward normals and inner medial axis. It should be clear that the results also hold for unbounded component, outward normals and outer medial axis. For a point $x \in \Sigma$, let m_x denote the center of the inner medial ball meeting Σ at x and ρ_x its radius.

It follows almost immediately from our sampling conditions that each point of Σ and a point not far away from Σ has a sample point nearby. Lemma 1 and Corollary 1 formalize this idea.

Lemma 1. *Any point $x \in \Sigma$ has a sample point within $\varepsilon_1 f(x)$ distance where $\varepsilon_1 = (\delta + \varepsilon + \delta\varepsilon)$.*

Since $f(x) \leq \rho_x$ for any point $x \in \Sigma$, the following corollary is immediate.

Corollary 1. *Any point $y \in \mathbb{R}^3$ with $\|y - \tilde{y}\| = \delta \rho_{\tilde{y}}$ has a sample point within $\varepsilon_2 \rho_{\tilde{y}}$ distance where $\varepsilon_2 = (2\delta + \varepsilon + \delta\varepsilon)$.*

3 Empty Balls

A ball is *empty* if its interior is empty of points from P. A main ingredient in our analysis will be the existence of large empty balls. They in turn lead to the existence of large Delaunay balls that circumscribe Delaunay tetrahedra in $\operatorname{Del} P$. The centers of such Delaunay balls which are also Voronoi vertices in $\operatorname{Vor} P$ play crucial roles in the algorithms for normal and feature estimations. In this section, we present two lemmas that assure the existence of large empty balls with certain conditions.

Lemma 2 below assures that for each point $x \in \Sigma$ there is a large empty ball of radius almost as large as (i) $f(x)$ and (ii) ρ_x. Notice the differences between the distances of these balls from x. Also, see Figure 3.

Lemma 2. *A ball $B_{m,r}$ is empty of sample points from P if either*

(i) $\tilde{m} = x$, $\|m - x\| = f(x)$ *and* $r = (1 - 3\delta)f(x)$, *or*
(ii) $m = m_x$ *and* $r = (1 - \delta)\rho_x$.

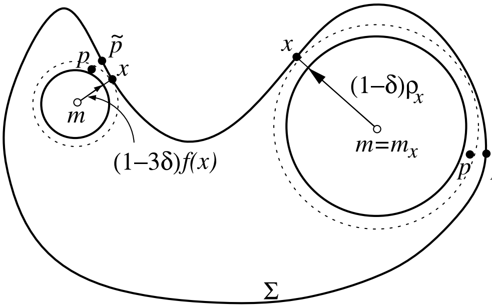


Fig. 3. Illustration for Lemma 2. The dotted big balls are not empty of sample points but their slightly shrunk copies (shown with solid boundaries)are.

Next, we show that, for each point x of Σ, there is a nearby large ball which is not only empty but also its boundary passes through a sample point close to x. Eventually these balls will be deformed to Delaunay balls for medial axis approximations.

Lemma 3. *For each point $x \in \Sigma$ there is an empty ball $B_{c,r}$ with $c \in \Omega_I$ that enjoys the following properties:*

(i) r *is at least* $(1 - 2\sqrt{\varepsilon_2})\rho_x$, m_x *is in* $B_{c,r}$ *and* $\|c - m_x\| \leq 2\sqrt{\varepsilon_2}\rho_x$ *where* ε_2 *defined in Corollary 1 is* $O(\varepsilon + \delta)$,
(ii) *its boundary contains a sample point* p *within a distance* $\varepsilon_3\rho_x$ *from* x *where* $\varepsilon_3 = 2{\varepsilon_2}^{\frac{1}{4}} + \delta$ *and* ε, δ *are sufficiently small.*

Observation: We could choose $\varepsilon_3 = O(\sqrt{\varepsilon_2}) = O(\sqrt{\varepsilon + \delta})$ though the radius of the empty ball becomes a constant fraction of ρ_x. Also, Lemma 3 remains valid when we replace ρ_x with $f(x)$.

4 Normal Approximation

We will approximate the normals by the vectors from the sample points toward the centers of the Delaunay balls incident to them. First, we derive an upper bound on this normal approximation error in Theorem 1. Then, we describe a simple algorithm for approximating the normals whose justification is given by the theorem and Lemma 3.

4.1 Analysis

The proof of Theorem 1 is based on the following idea. Consider tilting an empty ball whose boundary passes through a point p and whose center lies in the direction of $n_{\tilde{p}}$. The maximum amount of tilt with the constraint that the ball remains empty depends on how big the ball is and how close the sample points are.

Theorem 1. *Let $p \in P$ be incident to an empty ball $B_{c,r}$ where $r = \lambda f(\tilde{p})$ and $c \in \Omega_I$. Then,*

$$\sin(\angle \boldsymbol{pc}, \mathbf{n}_{\tilde{p}}) \leq 2(\frac{1}{\lambda} + 1)(\varepsilon_1 + 2\sqrt{\delta}) + O(\delta) + O(\varepsilon^2)$$

for a sufficiently small $\varepsilon > 0$ and $\delta > 0$.

Implications: Theorem 1 gives a general form of the normal approximation under a fairly general sampling assumption. One can derive different normal approximation bounds under different sampling assumptions from this general result. For example, if P is a $(\varepsilon, \varepsilon^2, -)$-sample we get an $O(\varepsilon)$ bound on the normal approximation error. In case P is a $(\varepsilon, \varepsilon, -)$-sample, we get an $O(\sqrt{\varepsilon})$ error bound. Another important implication is that Delaunay balls need not be too big to give good normal estimates. One can observe that if λ is only $\sqrt{\max\{\varepsilon, \delta\}}$, we get $O(\varepsilon^{\frac{1}{2}} + \delta^{\frac{1}{4}})$ error. Algorithmic implication of this fact is that a lot of sample points can qualify for normal estimation.

Theorem 1 remains valid even if the sample point p is replaced with any point $x \in \mathbb{R}^3$ meeting the conditions as stated in the corollary below. We use this fact later in feature estimation.

Corollary 2. *Let $x \in \mathbb{R}^3$ be any point with $\|x - \tilde{x}\| \leq \delta \rho_{\tilde{x}}$ and $B_{c,r}$ be any empty ball incident to x so that $r = \Omega(\rho_{\tilde{x}})$. Then, $\angle \boldsymbol{xc}, \mathbf{n}_{\tilde{x}} = O(\varepsilon + \sqrt{\delta})$ for sufficiently small ε and δ.*

4.2 Algorithm

We know from Theorem 1 that if there is a big Delaunay ball incident to a sample point p, then the vector from p to the center of the ball estimates the normal direction at the point $\tilde{p}$. On the other hand, the observation after the proof of Lemma 3 assures that for each point $x \in \Sigma$, there is a sample point p within $O(\sqrt{\varepsilon + \delta})f(x)$ distance with an empty ball of radius $\Omega(f(x))$. This means there is a big Delaunay ball incident to p where the vector $\boldsymbol{pc}$ approximates $\mathbf{n}_{\tilde{p}}$ and hence $\mathbf{n}_x$. Algorithmically we can exploit this fact by picking up sample points that are incident to big Delaunay balls only if we have a scale to measure 'big' Delaunay balls. For this we assume the third condition in the sampling which says that the sample is locally uniform.

Let λ_p be the distance of p to its κth nearest neighbor. By sampling condition $\lambda_p \geq \varepsilon f(\tilde{p})$. Therefore, any Delaunay ball incident to p with radius more than $\tau \lambda_p$ will give a normal estimation with an error $6(\frac{1}{\tau \varepsilon} + 1)(\varepsilon)$ according to Theorem 1 under the assumption that P is a $(\varepsilon, \varepsilon^2, \kappa)$-sample. It is important that λ_p is not arbitrarily large since then no Delaunay may qualify for the size threshold. This concern is alleviated by the fact that $\lambda_p \leq \varepsilon' f(\tilde{p})$ where $\varepsilon' = \left(\varepsilon + \frac{4\kappa+\varepsilon}{1-4\kappa\varepsilon}\right)\varepsilon$ [8].

Notice that the error decreases as τ increases. However, as we indicated before $\tau\varepsilon f(\tilde{p})$, the radius of the big Delaunay ball, can be as small as $\varepsilon^{\frac{1}{2}} f(\tilde{p})$ to give an $O(\sqrt{\varepsilon})$ error. This explains why a large number of Delaunay balls give good normal estimations as Figure 1 illustrates.

```
ApproximateNormal($P, \tau$)
    Compute Del $P$;
    for each $p \in P$ compute $\lambda_p$;
        if there is a Delaunay ball incident to $p$
        with radius larger than $\tau\lambda_p$
            Compute the largest Delaunay ball $B_{c,r}$
            incident to $p$;
            Approximate the normal direction at $p$ by $pc$.
        endif
```

Notice that, alternatively we could have eliminated the parameter τ in the algorithm by looking for the largest Delaunay ball incident to a set of k-nearest neighbors of p for some suitable k. Again, thanks to Lemma 3, we are assured that for a suitable k, one or more neighbors have Delaunay balls with radius almost equal to the medial balls. However, this approach limits the number of sample points where the normals are estimated. Because of our earlier observation, the normals can be estimated at more points where the Delaunay ball is big but not necessarily as big as the medial balls. In contrast, as we see next, feature estimation needs the Delaunay balls almost as big as the medial ones.

5 Feature Approximation

We approximate the local feature size at a sample point p by first approximating the medial axis with a set of discrete points and then measuring the distance of p from this set. We are guaranteed by Lemma 3 that there are many sample points which are incident to big Delaunay balls. The furthest Voronoi vertices from these sample points in Ω_I and Ω_O approximate the inner and outer medial axis respectively. For a point $p \in P$, we call the furthest Voronoi vertex from p in $V_p \cap \Omega_I$ as the inner pole p^+ of p. Similarly one may define the outer pole p^- of p which resides in Ω_O.

In line with the previous results on medial axis approximation [4,6,7], we claim that a certain subset of the medial axis is approximated by poles. Let x and x' be two points where the medial ball B centered at m meets Σ. Call $\angle xmx'$ the *medial angle* at m if it is the largest angle less than π made by any two such points of $B \cap \Sigma$. Let $M_\alpha \subseteq M$ be the subset where each point $m \in M_\alpha$ has a medial angle at least α.

5.1 Analysis

We show that each medial axis point m_x with a large enough medial angle is approximated by a pole. The idea is as follows. Consider the large ball incident to a sample point p guaranteed by Lemma 3. Then we deform it to a large Delaunay ball with the center at p^+. First, during this deformation the ball cannot be tilted too much since the vector from p to the center has to approximate the normal $\mathbf{n}_{\tilde{p}}$ by Theorem 1. Second, the center in the tilted direction cannot move too much due to Lemma 4 as stated below. The result of these constraints is that the center p^+ of the Delaunay ball remains close to the center of the original ball which in turn is close to m_x.

Lemma 4. *Let $B = B_{c,r}$ be an empty ball whose boundary passes through a sample point p. Let z be a point on Σ whose distance to the boundary of B is less than $\varepsilon' \rho_z$ for $\varepsilon' < 1$. Let $B' = B_{c',r'}$ be an empty ball obtained by expanding B while keeping c' on the ray $\mathbf{pc}$ and p on its boundary. If $\beta\rho_z \leq r \leq \rho_z$, then we have*

$$\|c - c'\| \leq \frac{(\varepsilon_1 + \varepsilon')(2 + \varepsilon')}{2\beta(1 - \cos\angle pcz) - 2\varepsilon_1 - 2\varepsilon' \cos\angle pcz}\rho_z.$$

Theorem 2. *For each point $m_x \in M_\alpha \cap \Omega_I$ where $\alpha = \varepsilon^{\frac{1}{4}} + \delta^{\frac{1}{4}}$, there is a sample point p within $O(\varepsilon^{\frac{1}{4}} + \delta^{\frac{1}{4}})\rho_x$ distance of x so that the pole p^+ lies within $O(\varepsilon^{\frac{1}{4}} + \delta^{\frac{1}{4}})\rho_x$ distance from m_x where ε and δ are sufficiently small.*

For each point $x \in \Sigma$ where $m_x \in M_\alpha$ the previous theorem guarantees the existence of a sample point p whose pole approximates m_x. Actually, the proof technique can also be used to show that any Delaunay ball with radius almost as big as ρ_x and incident to a sample point close to x has its center close to m_x.

Theorem 3. *Let $x \in \Sigma$ be a point so that $m_x \in M_\alpha$ for $\alpha = \varepsilon^{\frac{1}{4}} + \delta^{\frac{1}{4}}$. Then for any point $p \in P$ within $\varepsilon_3\rho_x$ distance of x and with an incident Delaunay ball of radius at least $(1 - O(\sqrt{\varepsilon} + \sqrt{\delta})\rho_x$, the pole p^+ lies within $O(\varepsilon^{\frac{1}{8}} + \delta^{\frac{1}{8}})\rho_x$ distance from m_x.*

5.2 Algorithm

Theorem 2 and Theorem 3 suggest the following algorithm for feature estimation at any point $x \in \Sigma$ where $m_x \in M_{\varepsilon^{\frac{1}{4}} + \delta^{\frac{1}{4}}}$. Theorem 2 says that x has a sample point p within a neighborhood of $\varepsilon_3\rho_x$ whose pole p^+ approximates m_x. Also, Theorem 3 says that *all* sample points within $\varepsilon_3\rho_x$ neighborhood of x with a large enough Delaunay ball have their poles approximate m_x. Therefore, if we take the pole of a sample point q whose distance to q is largest among all sample points within a neighborhood of x, we will get an approximation of m_x.

We search the neighborhood of x by taking k nearest neighbors of a sample point s close to x. If we assume that P is a $(\varepsilon, \delta, \kappa)$-sample for some $\kappa > 0$, κ-nearest neighbors cannot be arbitrarily close to x. Notice that if we do not prevent oversampling by the third condition of noisy sampling, we cannot make this assertion. In the algorithm, we simply allow an user supplied parameter k

to search the k nearest neighbors. Since we want to cover all points of Σ, we simply take all points of P and carry out the following computations.

For each point $p \in P$ we select k-nearest neighbors for a suitable k. Let N_p be this set of neighbors. First, for each $q \in N_p$, we determine the Voronoi vertex v_q in V_q which is furthest from q. This is one of the poles of q. Let $\ell_1(p) = \|v_q - q\|$. Select the point $p_1 \in N_p$ so that $\ell_1(p_1)$ is maximum among all points in N_p. By Theorem 2 and Theorem 3, v_{p_1} approximates a medial axis point m_x if $x \in M_{\varepsilon^{\frac{1}{4}}+\delta^{\frac{1}{4}}}$. However, we do not know if m_x is an inner medial axis point or an outer one. Without loss of generality assume that m_x is an inner medial axis point. To approximate the outer medial axis point for x, we determine the Voronoi vertex u_q in V_q for each $q \in N_p$ so that $\boldsymbol{qu_q}$ makes more than $\frac{\pi}{2}$ angle with $\boldsymbol{p_1v_{p_1}}$. Let $\ell_2(q) = \|u_q - q\|$. Then, we select the point $p_2 \in N_p$ so that $\ell_2(p_2)$ is maximum among all points in N_p. Again, appealing to Theorem 2 and Theorem 3 for outer medial axis, we can assert that u_{p_2} approximates a medial axis point for x.

```
ApproximateFeature($P, k$)
    Compute Del $P$; $L := \phi$;
    for each $p \in P$ compute $k$ nearest neighbors $N_p$;
        compute $p_1 \in N_p$ whose distance to
                its pole $v_{p_1}$ is maximum
                    among all points in $N_p$;
        compute $p_2 \in N_p$ with a pole $v_{p_2}$ so that
                $\angle \boldsymbol{p_2v_{p_2}}, \boldsymbol{p_1v_{p_1}} \geq \frac{\pi}{2}$ and $\|p_2 - v_{p_2}\|$
                is maximum among $N_p$;
        $L := L \cup \{v_{p_1}, v_{p_2}\}$;
    endfor
    for each $p \in P$ compute the distance of $p$ to $L$.
```

As we have observed already, a subset of the medial axis is not approximated by the poles. These are exactly the points on the medial axis which have a

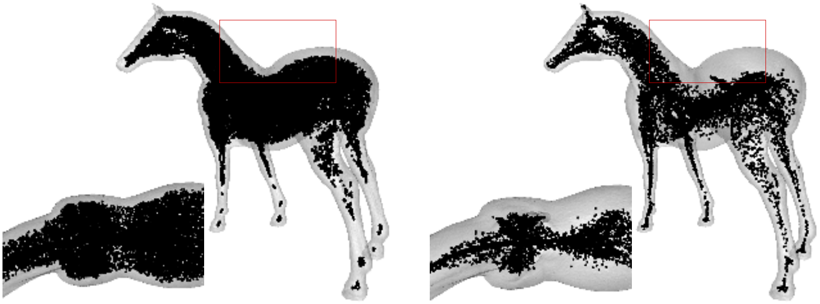


Fig. 4. Left: Medial axis approximated by centers of big Delaunay balls for a noisy Horse. For a chosen threshold, some parts of the legs do not have medial axis approximated though still many centers lie near the surface. Right: Medial axis well approximated by the poles as computed by ApproximateFeature.

small medial angle. This type of exclusions are also present in earlier medial axis approximation results [4,6,7]. The implication of this exclusion is that features cannot be properly estimated for points whose closest point on the medial axis resides in the excluded part. However, if the sampling is sufficiently dense, the excluded part is indeed small in most cases. Figure 4 shows the result of feature approximations for a model in three dimensions.

References

1. M. Alexa, J. Behr, D. Cohen-Or, S. Fleishman, D. Levin and C. Silva. Point set surfaces. *Proc. IEEE Visualization* (2001), 21–28.
2. N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discr. Comput. Geom.* **22** (1999), 481–504.
3. N. Amenta, M. Bern and D. Eppstein. The crust and the β-skeleton: combinatorial curve reconstruction. *Graphical Models and Image Processing*, **60** (1998), 125-135.
4. N. Amenta, S. Choi and R. K. Kolluri. The power crust, union of balls, and the medial axis transform. *Comput. Geom.: Theory Applications* **19** (2001), 127–153.
5. D. Attali and A. Montanvert. Modeling noise for a better simplification of skeletons. *Proc. Internat. Conf. Image Process.* **3**, 13–16, 1996.
6. J. D. Boissonnat and F. Cazals. Natural neighbor coordinates of points on a surface. *Comput. Geom. Theory Applications* (2001), 87–120.
7. F. Chazal and A. Lieutier. Stability and homotopy of a subset of the medial axis. *Proc. ACM Sympos. Solid Modeling and Applications* (2004), 243–248.
8. T. K. Dey and S. Goswami. Provable surface reconstruction from noisy samples. *Proc. 20th Annu. Sympos. Comput. Geom.* (2004), 330-339.
9. T. K. Dey and J. Sun. Adaptive MLS surfaces for reconstruction with guarantees. *Proc. Eurographics Sympos. Geom. Processing* (2005), 43-52.
10. T. K. Dey and W. Zhao. Approximating the medial axis from the Voronoi diagram with a convergence guarantee. *Algorithmica* **38** (2003), 179–200.
11. R. Kolluri. Provably good moving least squares. *Proc. ACM-SIAM Sympos. Discrete Algorithms* (2005), 1008-1017.
12. B. Mederos, N. Amenta, L. Velho and H. de Figueiredo. Surface reconstruction from noisy point clouds. *Proc. Eurographics Sympos. Geom. Processing* (2005), 53–62.
13. N. Mitra, J. Nguyen and L. Guibas. Estimating surface normals in noisy point cloud data. *Internat. J. Comput. Geom. & Applications* (2004).
14. M. Pauly, R. Keiser, L. Kobbelt and M. Gross. Shape modeling with point-sampled geometry. *Proc. ACM SIGGRAPH 2003*, 641–650.
15. T. K. Dey and J. Sun. Normal and Feature Approximations from Noisy Point Clouds. *Technical Report OSU-CISRC-4-05-TR26* (2005).
16. www.cse.ohio-state.edu/∼tamaldey/normfet.html

Coresets for Discrete Integration and Clustering[⋆]

Sariel Har-Peled[⋆⋆]

Department of Computer Science;
University of Illinois;
201 N. Goodwin Avenue;
Urbana, IL, 61801, USA

"The problem received the title of 'Buridan's sheep.' The biological code was taken from a young merino sheep, by the Casparo-Karpov method, at a moment when the sheep was between two feeding troughs full of mixed fodder. This code, along with additional data about sheep in general, was fed into CODD. The machine was required: a) to predict which trough the merino would choose, and b) to give the psychophysiological basis for this choice."
– The mystery of the hind leg, Arkady and Boris Strugatsky.

Abstract. Given a set P of n points on the real line and a (potentially infinite) family of functions, we investigate the problem of finding a small (weighted) subset $\mathcal{S} \subseteq P$, such that for any $f \in \mathcal{F}$, we have that $f(P)$ is a $(1 \pm \varepsilon)$-approximation to $f(\mathcal{S})$. Here, $f(Q) = \sum_{q \in Q} w(q) f(q)$ denotes the weighted discrete integral of f over the point set Q, where $w(q)$ is the weight assigned to the point q.

We study this problem, and provide tight bounds on the size $\mathcal{S}$ for several families of functions. As an application, we present some coreset constructions for clustering.

1 Introduction

Motivated by recent work on clustering, we investigate the following natural problem.

Problem 1. Let P be a set of points on the real line, and let $\mathcal{F}$ be a (potentially infinite) family of functions. A *ε-coreset for P* is a weighted subset $\mathcal{S} \subseteq P$, such that

$$\forall f \in \mathcal{F}, \quad f(P) \approx_\varepsilon f(\mathcal{S}),$$

where $f(P) = \sum_{p \in P} f(p)$, $f(\mathcal{S}) = \sum_{p \in \mathcal{S}} f(p) * w(p)$ and $w(p)$ is the associated weight of p in $\mathcal{S}$. (The notation $x \approx_\varepsilon y$ denotes the fact that $(1-\varepsilon)x \leq y \leq (1+\varepsilon)x$ and $(1-\varepsilon)y \leq x \leq (1+\varepsilon)y$.)

The problem is to find the smallest ε-coreset for P and $\mathcal{F}$. Note that such a coreset always exists as we can just take $\mathcal{S} = P$.

[⋆] The latest version of this paper is avaiable online [1].

[⋆⋆] Work on this paper was partially supported by a NSF CAREER award CCR-0132901.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 33–44, 2006.

If P is uniformly distributed on an interval on the real line, this is (very similar to) the standard problem of numerical integration on the real line studied in numerical-analysis. However, as our investigation demonstrates, this is fundamentally a different problem once the point set is not uniform. Similarly, there seems to be some indirect connection to discrepancy [2]. However, the author is unaware of any direct previous work on this problem.

To see how this problem naturally arises from clustering, consider the problem of computing *k-median clustering* (say, in the plane). For any set of centers C in the plane, every point of P is assigned the cost of being clustered using C; namely, the k-median clustering cost of P by C is the sum of distances of the points of P to their closest neighbor in C. Let $f_C(\cdot)$ the cost function induced by C, and let $\mathcal{F}_{kwc}$ be the set of all such functions. If one can find a small ε-coreset for P (for $\mathcal{F}_{kwc}$) then one can compute a good clustering of P directly on the (considerably smaller) coreset. Har-Peled and Kushal [3] showed that this problem, for a point set in $\mathbb{R}^d$, can be reduced to (a variant) of this one dimensional problem.

Coresets for k-median clustering are now relatively well understood, see [4,3,5]. See [6] for more details on the usage of coresets for clustering. However, if we are interested in handling more general clustering problems, like the centers being lines instead of points, we need to better understand the aforementioned more general problem. See the work by Feldman *et al.* [7] for preliminary results on this problem for the line clustering problem. For more information about clustering, see [8,9,10,11,12,13,14,15,16,17,18]. In particular, our work yields better coreset constructions for k-line median clustering (see Theorem 12) than what was known before [7].

Our approach is to systematically classify which families of functions have coresets and of what sizes, starting from (the trivial) family of linear functions and ending in the clustering functions mentioned above. The basic approach is quite natural, and has long history: We will partition the points into groups, and from each group pick one representative point (with weight equal to the group size). This is a classical technique used in computing estimates to summations and integrals (for example, bounding $\sum_{i=1}^{n} 1/i$ by partitioning the range $1, \ldots, n$ into the blocks $2^i, \ldots, 2^{i+1} - 1$, for $i = 1, \ldots, \lfloor \lg n \rfloor$). What makes our study (maybe) interesting, is that our partitions do not work just for a single function but for a family of functions, and require (especially towards the end) delicate and not completely trivial constructions. In particular, this gives rise to new partition schemes of point sets on the line (i.e., safe and secure partitions) which might be of independent interest.

The paper is organized as follows. In Section 2 we define some preliminary definitions. In Section 3, we study the problem for some simple families of functions. In Section 4, we introduce some partition schemes of point sets that are useful in constructing coresets for clustering functions. In Section 5, we use these partition scheme to construct coreset for the weighted k-median clustering problem on the line, and in Section 6, we extend this to handle the k-median function induced by k lines. We conclude in Section 7 with conclusions and some open problems.

2 Preliminaries

Two non-negative numbers x and y are $(1 \pm \varepsilon)$-*approximation* of each other if $(1-\varepsilon)x \le y \le (1+\varepsilon)x$ and $(1-\varepsilon)y \le x \le (1+\varepsilon)y$. We denote this fact by $x \approx_\varepsilon y$.

Observation 1. *Let x and y be two positive numbers and $\varepsilon < 1/4$. We have: (i) If $x \approx_\varepsilon y$ and $y \approx_\varepsilon z$ then $x \approx_{3\varepsilon} z$. (ii) If $|x-y| \le \varepsilon x$ then $x \approx_{2\varepsilon} y$. (iii) If $x \le (1+\varepsilon)y$ and $y \le (1+\varepsilon)x$ then $x \approx_\varepsilon y$.*

Two non-negative functions $f(\cdot)$ and $g(\cdot)$ are $(1 \pm \varepsilon)$-*approximation* of each other, denoted by $f \approx_\varepsilon g$, if $f(x) \approx_\varepsilon g(x)$, for all x.

3 Basic Coresets for Integration

In this section, we present coresets for several simple families of functions.

3.1 Linear Functions

Let $\mathcal{F}_{\text{linear}}$ be the set of affine functions of the form $f(x) = ax + b$. Then, clearly for the set P, the centroid point of P (i.e., the average value of P) with assigned weight $|P|$ is a coreset.

Lemma 1. *The family $\mathcal{F}_{\text{linear}}$ of linear functions have a coreset of size* 1.

3.2 Monotone Functions

Let $\mathcal{F}_{\text{dec}}$ (resp., $\mathcal{F}_{\text{inc}}$) be the *family of monotone decreasing non-negative functions* (resp., *family of monotone increasing non-negative functions*) from $\mathbb{R}$ to $\mathbb{R}^+$.

Lemma 2. *Given a set $P \subseteq \mathbb{R}$ of n numbers, one can compute an ε-coreset of P of size $O\left(\varepsilon^{-1} \log n\right)$ for the family of functions $\mathcal{F}_{\text{dec}}$.*

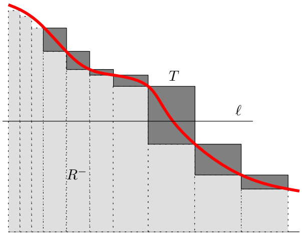
Proof. The following construction is somewhat of an overkill, but it would be useful later for other purposes.

The construction follows the exponential construction used in the 1-median coreset in [19]. Indeed, let z_i denote the ith point in the sorted order of the points of P. We partition P symmetrically into subsets, such that the size of the subsets increase in size as one comes toward the middle of the set. Formally, the set $A_i = \{z_i\}$ contains the ith point on the line, for $i = 1, \ldots, \mathsf{m}$, where $\mathsf{m} \ge 10/\varepsilon$ is a parameter to be determined shortly. Similarly, $R_i = \{z_{n-i+1}\}$, for $i = 1, \ldots, \mathsf{m}$. Set $\alpha_{\mathsf{m}} = \mathsf{m}$, and let $\alpha_{i+1} = \min\left(\lceil (1+\varepsilon/10)\alpha_i \rceil, n/2\right)$, for $i = \mathsf{m}, \ldots, \mathsf{m}'$, where $\alpha_{\mathsf{m}'}$ is the first number in this sequence equal to $n/2$. Now, let

$A_i = \{z_{\alpha_{i-1}+1}, \ldots, z_{\alpha_i}\}$ and $R_i = \{z_{n-\alpha_{i-1}}, \ldots, z_{n-\alpha_i+1}\}$, for $i = \mathsf{m}+1, \ldots, \mathsf{m}'$. See figure on the right. We will refer to a set A_i or R_i as a *chunk*. Consider the partition of P formed by the chunks $A_1, A_2, \ldots, A_{\mathsf{m}'}, B_{\mathsf{m}'}, \ldots, B_2, B_1$. To simplify the exposition, let $C_1, \ldots, C_{\mathsf{M}}$ denote the resulting sequence of sets, where $\mathsf{M} = 2\mathsf{m}'$. This is a partition of P into "exponential sets". The first/last m sets on the boundary are singletons, and all the other sets grow exponentially in cardinality, till they cover the whole set P.

Next, we pick an arbitrary point $l_i \in C_i$ and assign it weight $w_i = |C_i|$, for $i = 1, \ldots, \mathsf{M}$. Let $\mathcal{S}$ be the resulting weighted set of points. We claim that this is a coreset for any function of $\mathcal{F}_{\mathrm{dec}}$.

For the sake of analysis we can assume that $P = \{1, 2, \ldots, n\}$. Indeed, this can be realized by stretching and translating the real-axis appropriately and observing that the resulting function is still monotone decreasing. The claim now follows by a simple integration argument.



Indeed, let $f \in \mathcal{F}_{\mathrm{dec}}$ be an arbitrary function. For a chunk $C_i = \{j_i, \ldots, j_{i+1} - 1\}$, we observe that its contribution to $f(\mathcal{S})$ can be interpreted as a bar (i.e., rectangle) in a histogram based at the interval $\mathcal{I}_i = [j_i - 1, \ldots, j_{i+1} - 1]$ and having height $f(l_i)$, for $i = 1, \ldots, \mathsf{M}$. Consider the error zone formed by this rectangle, as it gets its maximum height (by setting $l_i = j_i$) and its minimum height (by setting $l_i = j_{i+1}-1$). Clearly, this zone of error is a rectangle $\mathrm{r}_i = \mathcal{I}_i \times [f(j_{i+1}-1)), f(j_i)]$. Let $T = \mathrm{r}_1 \cup \ldots \cup \mathrm{r}_{\mathsf{M}}$ be the resulting set formed by these of rectangles. Since $f(\cdot)$ is monotone decreasing, a horizontal line crosses the interior of at most one of the rectangle of T. Let R^- be the histogram $\bigcup_i \mathcal{I}_i \times f(j_{i+1} - 1)$, and let R^+ be the histogram $\bigcup_i \mathcal{I}_i \times f(j_i)$. Clearly, $R^- \subseteq R^+$, $R^+ = R^- \cup T$, $\mathrm{area}(R^+) = \mathrm{area}(R^-) + \mathrm{area}(T)$, $\mathrm{area}(R^-) \le f(\mathcal{S}) \le \mathrm{area}(R^+)$, and $\mathrm{area}(R^-) \le f(P) \le \mathrm{area}(R^+)$.

Furthermore, by construction, we have that

$$|\mathcal{I}_i| \le \frac{\varepsilon}{4} \sum_{j<i} |\mathcal{I}_j|,$$

for $i = \mathsf{m}+1, \ldots, \mathsf{M}$, where $|\mathcal{I}_i|$ denotes the length of this interval. In particular, this implies that for any horizontal line ℓ we have that $|T \cap \ell| \le (\varepsilon/4)\, |R^- \cap \ell|$. Now, imagine computing the area of T by integrating the length of the intersection of a horizontal line with T. We have that $\mathrm{area}(T) \le (\varepsilon/4)\mathrm{area}(R^-)$. This implies that

$$\begin{aligned}\mathcal{E} = |f(\mathcal{S}) - f(P)| &\le \mathrm{area}(R^+) - \mathrm{area}(R^-) = \mathrm{area}(T) \le (\varepsilon/4)\mathrm{area}(R^-)\\ &\le (\varepsilon/4)\min(f(P), f(\mathcal{S})),\end{aligned}$$

as required. □

Clearly, the same construction works for monotonically increasing function. In fact, the construction also works for a function which is decreasing and then increasing, as can be verified. In particular, let $\mathcal{F}_{\mathrm{dec}\to\mathrm{inc}}$ denote the set of non-negative functions which are first monotonically decreasing, and then they become monotonically increasing. We summarize:

Theorem 2. *Let P be a set of n points on the real line. One can construction a ε-coreset, of size $O(\varepsilon^{-1}\log n)$, that works for any function that belongs to $\mathcal{F}_{\mathrm{inc}} \cup \mathcal{F}_{\mathrm{dec}} \cup \mathcal{F}_{\mathrm{dec}\to\mathrm{inc}}$.*

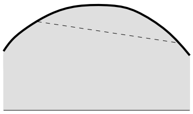
Since non-negative convex functions are also in $\mathcal{F}_{\mathrm{dec}\to\mathrm{inc}}$, it follows that this also holds for convex functions.

Let $\mathcal{F}_{\mathrm{inc}\to\mathrm{dec}}$ denote the set of non-negative functions which are monotonically increasing, and then they become monotonically decreasing afterwards.

Lemma 3. *Any ε-coreset for $\mathcal{F}_{\mathrm{inc}\to\mathrm{dec}}$ for a set P of n points on the real line, must include all points of P.*

Proof. In full version, see [1]. □

3.3 Concave Functions



Definition 3. A function $f : \mathbb{R} \to \mathbb{R}^+$ is *concave* on the interval $\mathcal{I}$, if for all $x, y \in \mathcal{I}$ and $\alpha \in [0,1]$ we have that $\alpha f(x) + (1-\alpha) f(y) \leq f(\alpha x + (1-\alpha) y)$.

Let $\mathcal{F}_{\mathrm{concave}}(\mathcal{I})$ denote the family of concave non-negative functions defined over the interval $\mathcal{I}$.

Definition 4. An interval $\mathcal{J}$ is *ε-oblivious* for $\mathcal{F}_{\mathrm{concave}}(\mathcal{I})$, if $\mathcal{J} \subseteq \mathcal{I}$ and for any $f \in \mathcal{F}_{\mathrm{concave}}(\mathcal{I})$ we have that $f(x) \approx_{\varepsilon/10} f(y)$, for all $x, y \in \mathcal{J}$.

Lemma 4. *Let $\mathcal{I} = [a,b]$, and consider $x \in [a,b]$. Then, there is an ε-oblivious interval, for $\mathcal{F}_{\mathrm{concave}}(\mathcal{I})$, centered at x of length $(\varepsilon/40)\min((x-a),(b-x))$.*

Proof. In full version, see [1]. □

Clearly, if $\mathcal{I}$ is a ε-oblivious interval, then we can pick an arbitrary point of $Q = \mathcal{I} \cap P$ and assign it weight equal to $|Q|$ as a representative of Q in the resulting coreset. Lemma 4 now implies that $f(Q)$ would be well approximated by this single point. The problem is that one can not cover a given interval by oblivious intervals, since an infinite number of such intervals is required.

Theorem 5. *Let P be a set of n points on the real line, let $\mathcal{I}$ be an interval containing P. Then one can compute a ε-coreset for $\mathcal{F}_{\mathrm{concave}}(\mathcal{I})$ of size $O(\varepsilon^{-1}\log n)$.*

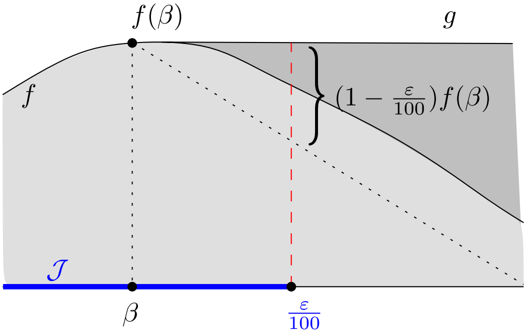
Proof. Assume $\mathcal{I} = [0,1]$. Tile the interval $[\varepsilon/100, 1-\varepsilon/100]$ by ε-oblivious intervals. The number of oblivious intervals required is $O(\varepsilon^{-1}\log(1/\varepsilon))$. Indeed, we start tiling from the middle, and let $x_1 = 1/2$. The ith ε-oblivious interval is $[x_{i+1}, x_i]$, where by Lemma 4 we can set $x_{i+1} = (1-\varepsilon/40)x_i$. Thus, for

$j > 2\lceil(40/\varepsilon)\rceil \ln(100/\varepsilon)$ intervals, we have that $x_j < \varepsilon/100$, as can be easily verified. We use the symmetric tiling for the interval $[1/2, 1-\varepsilon/100]$.

Compute for each such oblivious interval its coreset representative. As such, we are left with handling the margin intervals. Let $\mathcal{J} = [0, \varepsilon/100]$, and consider a function $f \in \mathcal{F}_{\text{concave}}(\mathcal{I})$.

The coreset we use for $Q = \mathcal{J} \cap P$ is the $(\varepsilon/30)$-coreset for monotone increasing functions of Theorem 2. Let $\mathcal{S}$ denote the resulting coreset for Q. We use, up to symmetry, the same coreset construction for $[1-\varepsilon/100, 1] \cap P$. Every oblivious interval contributes one point to the coreset. Overall, we have a coreset of size $O(\varepsilon^{-1}\log n)$. We remain with the task of proving that this subset is indeed the required ε-coreset.

Let $g(x) = \max_{0 \le y \le x} f(y)$. Note that $g(x) - f(x)$ is maximized in $\mathcal{J}$, if the maximum of $f(\cdot)$ lies at point β that is inside $\mathcal{J}$ and $x = \varepsilon/100$ is at the right endpoint of $\mathcal{J}$. But then, we have by convexity, that $f(x) \ge (1-\varepsilon/100)f(\beta)$. This implies that for any $x \in \mathcal{J}$, we have that $f(x) \le g(x) \le f(\beta) \le f(x)/(1-\varepsilon/100) \le (1+\varepsilon/50)f(x)$.



$f(\beta)$
g
f
$(1 - \frac{\varepsilon}{100})f(\beta)$
$\mathcal{J}$
β
$\frac{\varepsilon}{100}$

Thus, we can use $g(x)$ instead of $f(x)$ in $\mathcal{J}$, introducing a small $(\varepsilon/50)$-error in the process. Now, since the $(\varepsilon/10)$-coreset $\mathcal{S}$ for Q can handle monotone functions, it follows that it is a $(\varepsilon/30)$-coreset for $f(\cdot)$ on this interval. Formally,

$$f(Q) \le g(Q) \le (1+\varepsilon/30)g(\mathcal{S}) \le (1+\varepsilon/50)(1+\varepsilon/30)f(\mathcal{S}) \le (1+\varepsilon/20)f(\mathcal{S}),$$

and similarly,

$$f(Q) \ge \frac{1}{1+\varepsilon/50} g(Q) \ge \frac{1-\varepsilon/30}{1+\varepsilon/50} g(\mathcal{S}) \ge \frac{1-\varepsilon/30}{(1+\varepsilon/50)^2} f(\mathcal{S}) \ge (1-\varepsilon/10)f(\mathcal{S}).$$

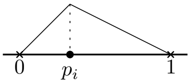
This implies that $\mathcal{S}$ is an ε-coreset for Q for the function f. Thus, collecting the two coresets of the margin intervals, and all the coresets for the oblivious intervals results in a ε-coreset for P. □

Somewhat surprisingly, the coreset constructed in Theorem 5 is of optimal size, up to a multiplicative constant.

Lemma 5. *In the worst case, any ε-coreset for $\mathcal{F}_{\text{concave}}(\mathcal{I})$ is of size $\Omega(\varepsilon^{-1}\log n)$.*

Proof. In full version, see [1]. □

Remark 1. Somewhat surprisingly the lower bound of Lemma 5 works even when the function $f_i(x)$ is a "triangle" function, namely, $f_i(x) = \min(x/p_i, (1-x)/(1-p_i))$. The same estimates as above hold with minor "noise" which do not effect the conclusion.



4 Partitioning Schemes

Let P be a set of n points on the real line. In this section, we investigate different ways of partitioning P into subsets, such that each subset has some kind of separation property from the rest of the point set. Intuitively, all the coreset constructions so far were based on partition schemes, and we need to have better understanding of such partitions to have more general coresets.

Definition 6 (Partition). A partition $\mathcal{P}$ of a point-set $P \subseteq \mathbb{R}$ is a set of disjoint subsets $S_1, \ldots, S_m$ of P, such that $\cup_i S_i = P$ and $\mathcal{B}(S_i) \cap \mathcal{B}(S_j) = \emptyset$, for $i \neq j$, where $\mathcal{B}(S)$ denotes is the smallest interval containing the set $S \subseteq \mathbb{R}$.

In the following, for an interval $\mathcal{I}$ and a positive real number c, we denote by $c\mathcal{I}$ the interval resulting from scaling $\mathcal{I}$ up by a factor of c around the middle point of $\mathcal{I}$.

For a partition $\mathcal{P}$ of P into disjoint subsets $\langle S_1, \ldots, S_m \rangle$ and a set $I \subseteq \mathbb{R}$, let

$$\mathcal{P} \setminus I = \left\{ S_i \,\middle|\, S_i \cap I = \emptyset \right\}$$

denote the family of sets of $\mathcal{P}$ that are completely outside I. Similarly, let

$$\mathcal{P} \cap I = \left\{ S_i \,\middle|\, S_i \subseteq I \right\}$$

denote the family of sets of $\mathcal{P}$ that are completely inside I. Note that $(\mathcal{P} \cap I) \cup (\mathcal{P} \setminus I)$ is not necessarily equal to $\mathcal{P}$, as some sets in $\mathcal{P}$ might intersect both I and $\mathbb{R} \setminus I$. We remind the reader that $\cup(\mathcal{P} \setminus I) = \cup_{S \in \mathcal{P} \setminus I} S$.

4.1 Safe Partitions

Definition 7 (ε-safe partition). A set $S \subseteq \mathbb{R}$ is *ε-safe* in relation to a set $P \subseteq \mathbb{R}$ if either $|S| = 1$ or alternatively $|P \setminus 3\mathcal{B}(S)| \geq |S|/\varepsilon$. Namely, there are a "lot" of points of P that are "faraway" from S.

As such, a partition $\mathcal{P}$ is *ε-safe* if either $|S_i| = 1$ or we have that $|\cup(\mathcal{P} \setminus 3\mathcal{B}(S_i))| \geq |S_i|/\varepsilon$, for $i = 1, \ldots, m$.

Lemma 6. *For a set P of n points on the real line, there exists an ε-safe partition of size $O(\varepsilon^{-1} \log n)$. And in the worst case, any ε-safe partition of P must be of size $\Omega(\varepsilon^{-1} \log n)$.*

Proof. In full version, see [1]. □

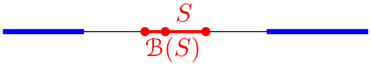


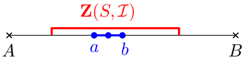
Fig. 1. The interval $\mathcal{B}(S)$ and its associated two safe regions. The safe regions must contain at least $|S|/\varepsilon$ points.

4.2 Secure Partitions

As reality had demonstrated repeatedly, there is no real safety without security (the reader mystified by this comment, should see Remark 2).

Definition 8 (ε-secure partition)

For an interval $\mathcal{I} = [A, B]$ and a set $S \subseteq \mathcal{I}$, the *security zone* for S, denoted by $\mathbf{Z}(S, \mathcal{I})$ is the interval $[(a + A)/2, (b + B)/2]$, where $\mathcal{B}(S) = [a, b]$.



Let P be a set of n points on the real line, and let $\mathcal{I} = [a, b]$ be a given interval that contains P. Informally, a partition $\mathcal{P} = \{S_1,. \ldots, S_m\}$ of P is *ε-secure* in $\mathcal{I}$, if we have that S_i is ε-safe in relation to the set $P \cap \mathbf{Z}(S_i, \mathcal{I})$, for all i. Formally, let $\mathcal{P}_i = \mathcal{P} \cap \mathbf{Z}(S_i, \mathcal{I})$ be the partition inside $\mathbf{Z}(S_i, \mathcal{I})$ induced by $\mathcal{P}$, and we require that S_i is ε-safe in relation to the partition $\mathcal{P}_i$. See Figure 2 for an alternative equivalent definition.

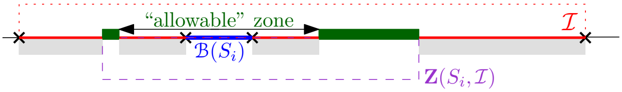


Fig. 2. The set S_i and its associated “allowable” zone $\mathbf{Z}(S_i, \mathcal{I}) \setminus 3\mathcal{B}(S_i)$. For $\mathcal{P}$ to be secure, the “allowable” zone of S_i must include sets of $\mathcal{P}$ with total mass exceeding $|S_i| / \varepsilon$. And this has to hold for all sets S_i in $\mathcal{P}$.

Intuitively, $\mathcal{P}$ is ε-secure if every subset in the partition has enough sets with sufficient total mass in the partition “faraway” from it, that are not too close to the endpoints of the host interval $\mathcal{I}$.

Lemma 7. *Let P be a set of n points on the real line contained inside the interval $\mathcal{I}$. Then, there exists a ε-secure partition of P in relation to $\mathcal{I}$ of size $O(\varepsilon^{-2} \log^2 n)$.*

Proof. In full version, see [1]. □

An improved construction.

Theorem 9. *Let P be a set of n points on the real line contained inside the interval $\mathcal{I}$. Then, there exists a ε-secure partition of P in relation to $\mathcal{I}$ of size $O(\varepsilon^{-1} \log n)$.*

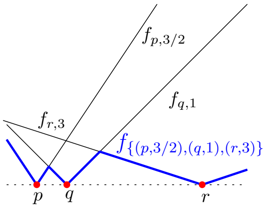
Proof. In full version, see [1]. □

5 Coreset for Weighted Centers

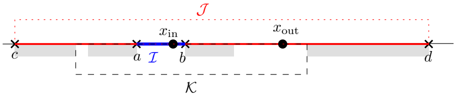
Let $\mathcal{F}_{kwc}$ be the family of functions induced by k weighted centers on the real line. Formally, for a point $p \in \mathbb{R}$ and weight $w \in \mathbb{R}^+$, let $f_{p,w}(x) = w * |x - p|$. Given a set W of k weighted points $(p_1, w_1), \ldots, (p_k, w_k)$, let $f_W(x) = \min_i f_{p_i, w_i}(x)$ denote the cost function of clustering x using its nearest weighted center in W, for all $x \in \mathbb{R}$.

Let $\widehat{W}$ denote the set of all possible k weighted points on the real line. Then, we have that

$$\mathcal{F}_{kwc} = \left\{ f_W(\cdot) \;\middle|\; W \in \widehat{W} \right\}.$$



Thus, for a point set $P \subseteq \mathbb{R}$ and $W \in \widehat{W}$, the quantity $f_W(P)$ is the price of k-median clustering of P by the weighted centers of W. We are now interested in computing a coreset for this family. We need the following technical lemma.



Lemma 8. *Let $\mathcal{I} = [a,b]$ and $\mathcal{J} = [c,d]$ be two intervals on the real line, such that $\mathcal{I} \subseteq \mathcal{J}$, and let $\mathcal{K} = [(a+c)/2, (b+d)/2]$. Let $f_W \in \mathcal{F}_{kwc}$ be a function, such that none of its centers are in the set $\mathcal{J} \setminus \mathcal{I}$ (i.e., $W \subseteq (\mathbb{R} \setminus \mathcal{J}) \cup \mathcal{I}$). Finally, let $x_{\text{out}} \in \mathcal{K} \setminus 3\mathcal{I}$ and $x_{\text{in}} \in \mathcal{I}$ be any two points. Then $f_W(x_{\text{in}}) \leq 3 f_W(x_{\text{out}})$.*

Proof. In full version, see [1]. □

Theorem 10. *Let P be a set of n points on the real line. There exists a ε-coreset for $\mathcal{F}_{kwc}$ of size $O\left(\left(\varepsilon^{-1}\log n\right)^{k+1}\right)$.*

Proof. In full version, see [1]. □

Remark 2. The reader might wanter why we need secure partitions for proving Theorem 10, and maybe safe partitions are enough. However, a careful inspection of the proof reveals that the inductive hypothesis in the proof fails if we use only safe partitions. Thus, we need the more involved and painful construction of secure partitions for our purposes.

5.1 A Lower Bound

Theorem 11. *A ε-coreset for a set of n points for the set of functions $\mathcal{F}_{kwc}$, has to be of size $\Omega\left(\max\left[(k/\varepsilon)\log(n/k)\,,\,2^k\right]\right)$ in the worst case.*

Proof. In full version, see [1]. □

6 Coreset for Weighted Lines

Consider a line ℓ in $\mathbb{R}^d$, for $d > 1$, and another line σ. For any point $p \in \sigma$, we are interested in its distance to ℓ. It is easy to verify that if we parameterize σ uniformly by a number $t \in \mathbb{R}$, then the *line distance function* $f(t) = \min_{x \in \ell} \|\sigma(t) - x\|$, has the form $f(t) = \sqrt{\gamma^2 + \beta^2(t-\alpha)^2}$, where $\beta \leq 1$ and $\gamma \geq 0$. Note, that $f(t)$ is non-negative, symmetric function around α, realizing its minimum at $t = \alpha$. Let $\mathcal{F}_{\text{dline}}$ denote the family of all such functions.

Lemma 9. *Let $f(t) = \sqrt{\gamma^2 + \beta^2(t-\alpha)^2}$ be a line distance function, and let x and y be two real numbers, such that $y < x < \alpha$ and $\alpha - y \leq \eta(\alpha - x)$, where $\eta \geq 1$ is a constant. Then, $f(y) \leq \eta * f(x)$.*

Let $\mathcal{F}_{k-\mathrm{lines}}$ be the function formed by the minimization diagram of k functions of $\mathcal{F}_{\mathrm{dline}}$. A function of such family has a natural interpretation as the distance of a point on a line, to the closest line in a set of k lines. Next, we prove the analogue of Lemma 8 for this family of functions.

Lemma 10. *Let $\mathcal{I} = [a,b]$ and $\mathcal{J} = [c,d]$ be two intervals on the real line, such that $\mathcal{I} \subseteq \mathcal{J}$, and let $\mathcal{K} = [(a+c)/2, (b+d)/2]$. Let $f \in \mathcal{F}_{k-\mathrm{lines}}$ be a function, such that all its minimums are either in $\mathcal{I}$ or outside $\mathcal{J}$. Finally, let $x_{\mathrm{out}} \in \mathcal{K} \setminus 3\mathcal{I}$ and $x_{\mathrm{in}} \in \mathcal{I}$ be any two points. Then $f(x_{\mathrm{in}}) \leq 3f(x_{\mathrm{out}})$.*

Proof. The function f is the minimization function of k functions $f_1, \ldots, f_k$. In particular, let F_O be the set of these functions with their minimum outside $\mathcal{J}$, and F_I be the set of these functions with minimums inside $\mathcal{I}$.

For a function $g \in F_I$, let $u \in \mathcal{I}$ be the location of its minimum. Consider the two points $x_1 = u - |x_{\mathrm{out}} - u|$ and $x_2 = u + |x_{\mathrm{out}} - u|$. Clearly, $\mathcal{I} \subseteq [x_1, x_2]$, and by the symmetry of g around u, we have that outside $[x_1, x_2]$ it is larger than it is inside this interval. As such, $g(x_{\mathrm{in}}) \leq g(x_1) = g(x_2) = g(x_{\mathrm{out}})$.

For a function $h \in F_O$, assume that x_{out} is to the right of $\mathcal{I}$. If the minimum of h is to the left of $\mathcal{J}$, then we are done as h is increasing on the interval $\mathcal{J}$, and as such $h(x_{\mathrm{out}}) \geq h(x_{\mathrm{in}})$. If the minimum of h is to the right of $\mathcal{J}$, as depicted on the right, then arguing as in Lemma 8 we have that $|x_{\mathrm{out}} - b| > |\mathcal{I}|$. Also, we have that $|x_{\mathrm{out}} - d| \geq |x_{\mathrm{out}} - b|$. Thus, by Lemma 9, we have that

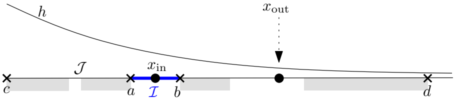


Fig. 3. The gray areas represent the "forbidden/insecure" areas for x_{out}

$$
\begin{aligned}
h(x_{\mathrm{out}}) &\geq \frac{d - x_{\mathrm{out}}}{d - x_{\mathrm{in}}} f_M(x_{\mathrm{in}}) \geq \frac{d - x_{\mathrm{out}}}{|\mathcal{I}| + |b - x_{\mathrm{out}}| + |x_{\mathrm{out}} - d|} f_M(x_{\mathrm{in}}) \\
&\geq \frac{d - x_{\mathrm{out}}}{3\,|x_{\mathrm{out}} - d|} f_M(x_{\mathrm{in}}) = \frac{f_M(x_{\mathrm{in}})}{3},
\end{aligned}
$$

as claimed. □

To construct a coreset for $\mathcal{F}_{k-\mathrm{lines}}$ using a construction similar to Theorem 10, we need to be able to handle the base case $k = 0$. Fortunately, the functions induced in such a case are "almost" concave.

Lemma 11. *Let $\mathcal{I} = [a,b]$ be an interval, and $P \subseteq \mathcal{I}$ be a set of n points. Then one can compute a ε-coreset $\mathcal{S}$, of size $O(\varepsilon^{-1} \log n)$, for the set of functions of $\mathcal{F}_{k-\mathrm{lines}}$ with minimums outside $\mathcal{I}$.*

Formally, the coreset works for functions f, such that $f \in \mathcal{F}_{k-\mathrm{lines}}$ is the minimization diagram of k line distance functions $f_1, \ldots, f_k \in \mathcal{F}_{\mathrm{dline}}$, and all of them having minimums outside $\mathcal{I}$.

Proof. In full version, see [1]. □

Theorem 12. *Let P be a set of n points on the real line. There exists a ε-coreset for $\mathcal{F}_{k-\text{lines}}$ of size $O\left(\left(\varepsilon^{-1}\log n\right)^{k+1}\right)$.*

Proof. In full version, see [1]. □

Remark 3. Theorem 12 is a substantial improvement over the result of Fiat *et al.* [7] that had coreset of size $\frac{2^{O(k^2)}}{\varepsilon^{2k+1}}\log^{4k-3} n$, for this problem. Also, our construction is arguably is simpler and more intuitive (well, at least for the author).

6.1 Applications

The results above immediately imply that there exists a k-line median coreset for clustering of small size. Namely, given a set P of n points in $\mathbb{R}^d$ one can find a small coreset of small size such that finding the k-lines of minimum median price (i.e., every point pays its distance to the closest line in the set k lines that serves as centers).

Theorem 13. *Given a set P of n points in $\mathbb{R}^d$, and a $k > 0$, there exists a ε-coreset for P for the problem of k-line median clustering. The size of the coreset is $O\left(k\varepsilon^{-k-d}\log^{k+2} n\right)$.*

Proof. In full version, see [1]. □

Plugging our construction into the standard machinery of random sampling leads to an efficient clustering algorithm, which computes $(1+\varepsilon)$-approximate k-median line clustering in near linear time. We omit any further details, see [7].

7 Conclusions

We had introduced the problem of computing coreset for discrete integration, and showed some tight coreset constructions for this problem, for various families of functions. We also used it to improve the coreset size for the problem of clustering points in $\mathbb{R}^d$ for the k-median line clustering.

In particular, we showed a coreset of size (roughly) $O(\log^{k+2} n)$ and a lower bound of 2^k, see Theorem 11. Previously, no non-trivial lower bound was known for this problem. Although there is still a gap between the upper and lower bound it is relatively "small", and we leave the improvement of both bounds as an open problem for further research. The lower bound also leaves open the question of how to cluster efficiently a set of n points with $k = \Omega(\log n)$ line centers. The lower bound implies that coresets can not help us in solving this problem efficiently, but maybe other techniques might work here. We leave this as an open problem for further research.

Another open problem is to extend the study of the discrete integration problem to dimensions higher than one.

Acknowledgments

The author would like to thank Danny Feldman for useful and insightful discussions on the problems studied in this paper.

References

1. Har-Peled, S.: Coresets for discrete integration and clustering. Available from `http://www.uiuc.edu/~sariel/papers/06/integrate` (2006)
2. Matoušek, J.: Geometric Discrepancy. Springer (1999)
3. Har-Peled, S., Kushal, A.: Smaller coresets for k-median and k-means clustering. In: Proc. 21st Annu. ACM Sympos. Comput. Geom. (2005) 126–134
4. Har-Peled, S., Mazumdar, S.: Coresets for k-means and k-median clustering and their applications. In: Proc. 36th Annu. ACM Sympos. Theory Comput. (2004) 291–300
5. Chen, K.: On k-median clustering in high dimensions. In: Proc. 17th ACM-SIAM Sympos. Discrete Algorithms. (2006) 1177–1185
6. Agarwal, P.K., Har-Peled, S., Varadarajan, K.: Geometric approximation via coresets. In Goodman, J.E., Pach, J., Welzl, E., eds.: Combinatorial and Computational Geometry. Math. Sci. Research Inst. Pub. Cambridge (2005)
7. Feldman, D., Fiat, A., Sharir, M.: Coresets for weighted facilities and their applications. manucript (2006)
8. Alon, N., Dar, S., Parnas, M., Ron, D.: Testing of clustering. In: Proc. 41th Annu. IEEE Sympos. Found. Comput. Sci. (2000) 240–250
9. Agarwal, P.K., Procopiuc, C.M.: Approximation algorithms for projective clustering. In: Proc. 11th ACM-SIAM Sympos. Discrete Algorithms. (2000) 538–547
10. Bădoiu, M., Clarkson, K.: Smaller coresets for balls. In: Proc. 14th ACM-SIAM Sympos. Discrete Algorithms. (2003) 801–802
11. Eppstein, D.: Fast hierarchical clustering and other applications of dynamic closest pairs. In: Proc. 9th ACM-SIAM Sympos. Discrete Algorithms. (1998) 619–628
12. Feder, T., Greene, D.H.: Optimal algorithms for approximate clustering. In: Proc. 20th Annu. ACM Sympos. Theory Comput. (1988) 434–444
13. Gonzalez, T.: Clustering to minimize the maximum intercluster distance. Theoret. Comput. Sci. **38** (1985) 293–306
14. Indyk, P.: A sublinear time approximation scheme for clustering in metric spaces. In: Proc. 40th Annu. IEEE Sympos. Found. Comput. Sci. (1999) 100-110.
15. Mishra, N., Oblinger, D., Pitt, L.: Sublinear time approximate clustering. In: Proc. 12th ACM-SIAM Sympos. Discrete Algorithms. (2001) 439–447
16. Ostrovsky, R., Rabani, Y.: Polynomial time approximation schemes for geometric k-clustering. In: Proc. 41st Symp. Foundations of Computer Science, IEEE (2000) 349–358
17. Kanungo, T., Mount, D.M., Netanyahu, N.S., Piatko, C.D., Silverman, R., Wu, A.Y.: A local search approximation algorithm for k-means clustering. Comput. Geom. Theory Appl. **28** (2004) 89–112
18. Inaba, M., Katoh, N., Imai, H.: Applications of weighted voronoi diagrams and randomization to variance-based k-clustering. In: Proc. 10th Annu. ACM Sympos. Comput. Geom. (1994) 332–339
19. Har-Peled, S.: How to get close to the median shape. In: Proc. 22nd Annu. ACM Sympos. Comput. Geom. (2006) To appear. Available from `http://www.uiuc.edu/~sariel/papers/05/l1_fitting/`.

Self-assemblying Classes of Shapes with a Minimum Number of Tiles, and in Optimal Time*

Florent Becker[1], Ivan Rapaport[2], and Éric Rémila[1]

[1] Laboratoire de l'Informatique du Parallélisme,
UMR 5668 CNRS-INRIA-Univ. Lyon 1-ENS Lyon, France
{florent.becker, eric.remila}@ens-lyon.fr

[2] Departamento de Ingeniería Matemática and
Centro de Modelamiento Matemático
UMR 2071-CNRS, Universidad de Chile
irapapor@dim.uchile.cl

Abstract. In this paper we construct fixed finite tile systems that assemble into particular classes of shapes. Moreover, given an arbitrary n, we show how to calculate the tile concentrations in order to ensure that the expected size of the produced shape is n. For rectangles and squares our constructions are optimal (with respect to the size of the systems). We also introduce the notion of *parallel time*, which is a good approximation of the classical asynchronous time. We prove that our tile systems produce the rectangles and squares in linear parallel time (with respect to the diameter). Those results are optimal. Finally, we introduce the class of diamonds. For these shapes we construct a non trivial tile system having also a linear parallel time complexity.

1 Introduction

The tile assembly model was introduced by Rothemund and Winfree [5,7]. This model, based on the classical one of Wang [6], includes a mechanism of growth (a dynamics) which takes into account global parameters such as the temperature and the tile concentrations.

The individual components are square tiles. These tiles "float" on the two dimensional plane. They can not be rotated. Each side of a tile has a specific "glue". When two tiles collide they stick if their abbuting sides have the same glue and, crucially, if the strength of the glue is "high enough" with respect to the temperature.

The dynamics of such a tile system is modeled as a Markov process. The precise process we consider here was introduced by Adleman et al. [1]. It is, however, a simplification of the reversible version proposed by Winfree [7]. Roughly speaking, the higher the concentration of a particular tile the higher the rate at which

* This work was supported by programs Ecos-Conicyt, Conicyt "Anillo en Redes", and Fondap on Applied Mathematics.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 45–56, 2006.

it is encountered. And, when encountered, the particular tile can eventually be incorporated into the growing structure. At the end of the process, which begins with a "seed" tile placed at the origin of the plane, a given shape S will be produced. Aggarwal et al. [2] proved that the minimal number of tiles that uniquely produces the $m \times n$ rectangle is $\Omega(\frac{n^{\frac{1}{m}}}{m})$ if $m \ll n$ and $\Theta(\frac{\log n}{\log \log n})$ otherwise. In [1] it is shown that the average time complexity for uniquely producing an $n \times n$ square is $\Theta(n)$.

Of course, besides the assembly time, some other random variables are also relevant. In fact, in this work we focus our attention on the random variable that corresponds to the "size" of the produced shape. We therefore explore a new direction by searching for tile systems producing languages of shapes. We tackle the problem of producing three natural languages of shapes: squares, rectangles and diamonds.

Let us consider, for instance, the class of all squares. In our construction we fix the tile system in such a way that each time we run the Markov chain a (different) square is produced. Let us call N the random variable corresponding to the size of the produced square. If n is a fixed positive integer, then we will show how to calculate the tile concentrations in order to ensure that $\mathbb{E}(N) = n$. To our knowledge, the tile concentrations, a parameter of the original model, has never been seriously considered before. In other words, it is therefore not necessary to construct the tiles (which in fact are model representations of tiny molecules) for each shape we are asked to assemble.

We construct tile systems for rectangles and squares. Each of them turn out to be optimal with respect to its size. On the other hand we introduce the notion of parallel time. It is defined as the time needed to assemble a shape when all possible transitions are performed at once. Our constructions for squares and rectangles are also optimal in terms of the parallel time. In fact we get, with respect to the diameter of the shapes, a linear parallel time. We also show that the parallel time gives lower and upper bounds for the average time of the Markov process.

Finally we introduce the class of diamonds. A first approach gives a tile system with quadratic parallel time complexity (linear with respect to the surface). Nevertheless, by using a "firing squad" approach, we get the optimal linear parallel time complexity (linear with respect to the diameter).

2 Tile Systems

A tile system is a 5-tuple $\mathbf{T} =< T, t_0, \tau, g, P >$. Each of these variables is defined in the following.

The set of tiles. T is a finite set of tiles. Each of these tiles is an oriented unit square with the north, east, south and west edges labeled from some alphabet Σ of glues (or colors). For each $t \in T$, the labels of its four edges are canonically denoted $\sigma_N(t)$, $\sigma_E(t)$, $\sigma_S(t)$ and $\sigma_W(t)$.

The seed. $t_0 \in T$ is a particular tile known as the seed.

The temperature. τ is a positive integer called the temperature.

The strength function. The (glue) strength function g goes from $\Sigma \times \Sigma$ to $\mathbb{N}$. We assume that $g(\alpha, \beta) = 0$ for all α, β in Σ such that $\alpha \neq \beta$. The value $g(\alpha, \alpha)$ is called the strength of α. We also assume that the set of glues Σ contains a special one, denoted by *null*, such that for all α in Σ, $g(null, \alpha) = 0$. The tiles are represented as in Figure 1: the number of lines in front of the glue corresponds to the strength of it. There is one exception to that convention: no lines also mean strength 1.

The concentration. The concentration P associates to each tile $t \in T$ a positive value $P(t)$. The concentration function P satisfies $\Sigma_{t \in T} P(t) = 1$.

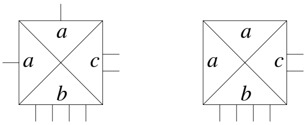


Fig. 1. Two ways of representing the same tile

In order to define the asynchronous and parallel dynamics we need to introduce the following concepts.

T-transitions. A configuration is a map from $\mathbb{Z}^2$ to $(T \cup \{empty\})$, where the tile *empty* is the one having in its four sides the glue *null*. Let A and B be two configurations. Suppose that there exist $t \in T$ and $(x, y) \in \mathbb{Z}^2$ such that $A = B$ except for (x, y) with $A(x, y) = empty$ and $B(x, y) = t$. If also

$$g(\sigma_E(t), \sigma_W(A(x+1, y)) + g(\sigma_W(t), \sigma_E(A(x-1, y))+$$

$$g(\sigma_N(t), \sigma_S(A(x, y+1)) + g(\sigma_S(t), \sigma_N(A(x, y-1)) \geq \tau$$

then we say that the position (x, y) is attachable in A, and we write $A \rightarrow_z B$ with $z = (t, (x, y))$. We write $A \rightarrow_{\mathbf{T}} B$ when such a z exists. Informally, this means that B can be obtained from A by adding a tile t in such a way that the total strength of the interaction between A and t is at least τ. Let $\rightarrow^*_{\mathbf{T}}$ denote the transitive closure of $\rightarrow_{\mathbf{T}}$.

Derived supertiles. The seed configuration, Γ_{t_0}, is the one that satisfies Γ_{t_0} $(0, 0) = t_0$ and, for all $(x, y) \neq (0, 0)$, $\Gamma_{t_0}(x, y) = empty$. The derived supertiles of the tile system $\mathbf{T}$ are those configurations X such that $\Gamma_{t_0} \rightarrow^*_{\mathbf{T}} X$.

Production of shapes. A shape is a 4-connected finite subset of $\mathbb{Z}^2$. The shape of a derived supertile A will be denoted by $[A]$ and corresponds to $\{(x, y) \in \mathbb{Z}^2 : A(x, y) \neq empty\}$.

2.1 The Asynchronous Dynamics

The dynamics of the tile system $\mathbf{T}$ is modeled as a continous time Markov process where the states are in one-to-one correspondence with the derived supertiles and the initial state corresponds to the seed coniguration Γ_{t_0}. There is a transition from state A to state B if $A \rightarrow_{\mathbf{T}} B$. If B is obtained from A by adding the tile t, then the rate of the transition is $P(t)$. More precisely, it is assumed that the time for the occurence of such a transition follows an exponential law of parameter $P(t)$, and, consequently, the average time necessary to make this transition is $1/P(t)$.

Suppose that in state A there are k possible transitions to states $B_1, \ldots, B_k$. And suppose that the transition rates are $P_1, \ldots, P_k$. Then, the probability to jump to state B_i equals $\frac{P_i}{P_1+\ldots+P_k}$. Finally, the time spent in state A follows an exponential law of parameter $P_1 + \ldots + P_k$.

A derived supertile A is called terminal if it is a sink state of the Markov process. In other words, if there is no supertile B such that $A \rightarrow_{\mathbf{T}} B$. The set of shapes produced by the tile system is $S(\mathbf{T}) = \{[A] : A \text{ is terminal}\}$.

Let $\mathcal{C}$ be a set of shapes. We say that the tile system $\mathbf{T}$ uniquely produces the set $\mathcal{C}$ if on one hand $S(\mathbf{T}) = \mathcal{C}$ and, on the other hand, the event "the structure grows indefinitely" has probability zero of ocurrence. This notion is a natural generalization of the one of Winfree where the set $\mathcal{C}$ was a singleton.

2.2 The Parallel Dynamics

Here, at each step, all possible transitions are performed at once. This parallel dynamics is deterministic. Sometimes it is easy to compute and it allows us to obtain bounds for the assembly time of the asynchronous model.

Let A be a derived supertile, and let $Trans$ be a set of transitions. $Trans$ is a set of independent transitions if, for any $z, z' \in Trans$ such that $A \rightarrow_z A_z$ and $A \rightarrow_{z'} A_{z'}$, z' is a possible transition from A_z and z is a possible transition from $A_{z'}$.

There can be several maximal sets of independent transitions from a given supertile. Nevertheless, given a terminal supertile F and a derived supertile A, one can define a unique $\parallel$ $\mathbf{T}$-transition from A to a A' such that F can be derived from A'. This parallel transition is given by taking the set of all attachable positions in A, and attaching to each of them the corresponding tile of F, when it is already attachable: let $Trans = \{t_1, \ldots, t_k\}$ be a maximal set of independent transitions from A which are compatible with F. Let A_k be such that $A \rightarrow_{t_1} A_1 \ldots A_{k-1} \rightarrow_{t_k} A_k$. We say that there is a parallel transition between A and A_k, and we note $A \rightarrow_{\parallel\mathbf{T}} A_k$.

Thus, the sequence of parallel transitions by which F is built is unique.

The *parallel time* π_A to assemble a derived supertile A is the number of parallel transitions needed to produce A. We will see that the parallel time is closely related to τ_A, the expected (asynchronous) assembly time of A. The *parallel instant* of a position (x, y) is the number of the parallel transition that puts a tile at (x, y).

This notion of parallel time is the adequate notion to compute the time efficiency of tilesets, as it is easier to compute than the expected time to complete the Markov process, but is a good approximation.

Proposition 1. *For every tile system $\boldsymbol{T}$ and every production P of $\boldsymbol{T}$, we have $(1/\chi_{max})\pi_P \le \tau_P$, and $\tau_P = O(\pi_P/\chi_{min})$, where χ_{min} and χ_{max} are the minimal and maximal tile concentrations. When all the concentrations are $1/k$, with k the number of tiles in $\boldsymbol{T}$, we have $\tau_P = \Theta(\pi_P k)$.*

We only give an idea of the proof, which is an extension of the proof of Adleman and Cheng[1]. They stated a version of this prosposition for cases where there is an order of dependency between the positions. But this order does not exist in every case; when it does not exist, we take the order induced by the parallel instants. This order has the properties of fairness that allow us to prove the proposition: any order that is compatible with the assembly contains a chain of length at least π_p.

3 Rectangles and Squares

Let m, n be positive integers. Let $R_{m,n} = \{(x, y) \,|\, 0 \le x < m\,,\, 0 \le y < n\}$ be the rectangle of width m and height n. Let us fix the temperature $\tau = 2$. With this temperature, by generalyzing the result of [1], it has been proved in [2] that the minimal number of tiles that uniquely produce the rectangle $R_{m,n}$ is $\Omega(\frac{n^{\frac{1}{m}}}{m})$ if $m \ll n$ and $\Theta(\frac{\log n}{\log\log n})$ otherwise.

Let us fix the set of tiles that appears in Figure 2. Let us consider t_{SW} as the seed. It is easy to notice that this set of tiles uniquely produces rectangles. If A, B, C are arbitrary positive values satisfying $A + B + C = 1$, then we fix the concentrations as follows:

$$\begin{array}{lll} P(t_S) = \frac{A(m-2)}{m-1} & P(t_{SE}) = \frac{A}{m-1} & P(t_W) = \frac{B(n-2)}{n-1} \\ P(t_{NW}) = \frac{B}{n-1} & P(t_\beta) = C & P(t_{SW}) = 0. \end{array}$$

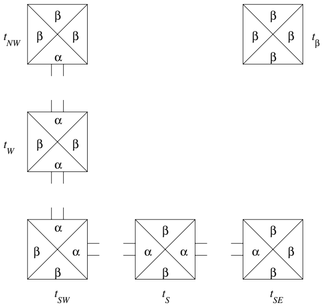


Fig. 2. The set of tiles used for producing rectangles

If we want to produce squares it is rather natural to create diagonals. Informally, we use three tiles in order to construct the diagonal. These three tiles -$t_D, t_{D_{down}}, t_{D_{up}}$- appear in Figure 3. We need two more tiles in order to fill the square: t_{β_2} for the northwest half and t_{β_1} for the southeast half. We need to have

$\beta_2 \neq \beta_1$, in order to avoid final productions which are not squares. The seed is t_D and the temperature is $\tau = 2$.

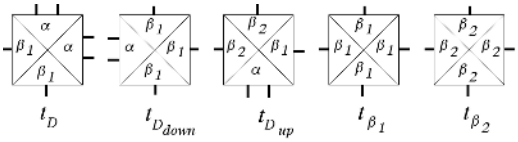


Fig. 3. The set of tiles used for producing squares

For arbitrary positive values A, B, C, D such that $A + B + C + D = 1$, we fix the concentrations as follows:

$$P(t_D) = \frac{A(n-2)}{n-1},\ P(t_{\beta_1}) = \frac{A}{n-1},\ P(t_{D_{down}}) = B,\ P(t_{D_{up}}) = C,\ P(t_{\beta_2}) = D$$

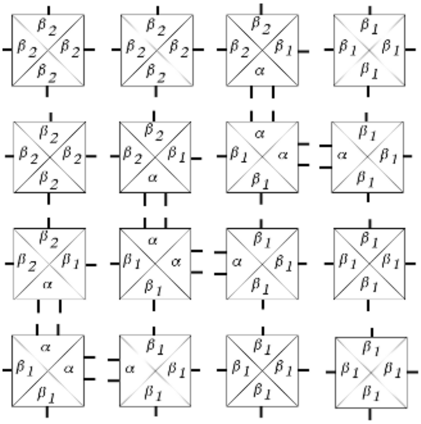


Fig. 4. An example of square production

Proposition 2. *The tile systems defined above uniquely produce rectangles and squares. In the case of rectangles, if M is the random variable corresponding to the width and N the random variable corresponding to the height, then $\mathbb{E}(M) = m$ and $\mathbb{E}(N) = n$. In the case of squares, if N is the random variable corresponding to the length of the side then $\mathbb{E}(N) = n$.*

Proof. Just for M in the case of rectangles. The other cases are very similar. Let $\epsilon = (m-1)^{-1}$. The random variable M follows a geometric law. More precisely, $\Pr\{M = k\} = (1-\epsilon)^{k-2}\epsilon$. So $\mathbb{E}(M) = \epsilon^{-1} + 1 = m$.

Proposition 3. *The tile systems which we defined above are the smallest ones that uniquely produce the set of all rectangles and the set of all squares. In the case of rectangles, each set of tiles whose set of final productions is formed by the $m \times n$ rectangles, with $m \geq 2$ and $n \geq 2$, contains at least six tiles. In the case of squares, each set of tiles whose set of final productions is formed by the $n \times n$ squares, with $n \geq 2$, contains at least five tiles.*

Proof. We just need to focus on glues of strength 2. With respect to that property there are 16 tiles.

Squares. Let us consider the set of all possible tilings of 2 × 2 squares. We have 8 cases (see Figure 5). Since we are assuming the seed to be in the leftmost position of the bottom the number of cases can be reduced to 5 (up to symmetry 1=3, 2=4, 6=7). In each case, we have at least three or four tiles. More precisely, 2 of cardinality three and 3 of cardinality four.

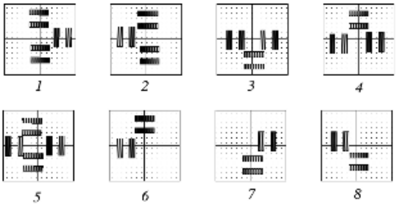


Fig. 5. The possible tilings of the 2 × 2 square

We want to prove that four tiles are not enough to generate squares. We have to test sets of four tiles which contain one of the previous sets which produce 2 × 2 squares. Each of the two cases of cardinality 3 may be completed with 13 tiles. Therefore, there are at most $2 \times 13 + 3 = 29$ cases to test. We first try to produce a 3×3 square with each set of tiles. In any exploration, we can easily see the impossibility for all the completions with the exception of the set numbered by 8.

For this latter case, we study the only (hypothetical) possible tiling of a 4×4 square obtained by self-assembly. Let t be the tile which has no glue of strength 2. Notice that t must be placed at the lower right corner and at the upper left corner of the square. Moreover, the opposite glues of t must be equal (in order to assemble any k-square with $k > 2$). Let t' be the tile with glue of strength 2 in its southern side. A t-tile of the left upper part of the square must touch a t'-tile below and another t-tile must touch a t'-tile in its right. Therefore, a t'-tile could be placed in the lower right corner of the square. A contradiction.

Rectangles. For rectangles we consider set of tiles which can produce the 2 × 2 square (i.e. containing a set of tiles obtained in Figure 5), the 3 × 2 rectangle, 2 × 3 rectangle.

We first assume that the only glue of strength 2 of the seed is on its northern side. In this case, we have 12 possible tilings of the 3 × 2 rectangle induced by successive productions, 7 with six tiles, 4 with five tiles, and 1 with four tiles. An obvious analysis proves that sets of 5 tiles obtained are not sufficient to tile the 2 × 3 rectangle.

For the set of four tiles, we can remark that another tile is necessary to tile the 2 × 3 rectangle and, furthermore, there exists two tiles with only a glue of strength 2 on their northern side which have the same strength 2 glue. Otherwise, some productions which are not in the upper right quarter of plane appear.

The case when the only glue of strength 2 of the seed is on its eastern side can be treated in a symmetric way. We now assume that the seed has two sides (the northern and the eastern one) with glues of strength 2. We study tilings of the 3 × 2 rectangle induced by successive productions We have two subcases according to the glues of strength 2 of the tile in position $(0, 1)$.

- When this tile has two sides (the southern and the eastern one) with glues of strength 2, we also have 12 cases: 7 with six tiles, 4 with five tiles, and 1 with four tiles. An obvious analysis proves that the sets of 5 tiles obtained are not sufficient to tile the 2 × 3 rectangle.
- When this tile has only one side (the southern one) with a glue of strength 2, we also have 12 cases (symmetric to the case when the seed has a unique side with glue of strength 2): 7 with six tiles, 4 with five tiles, and 1 with four tiles. An obvious analysis proves that the sets of 5 tiles obtained are not sufficient to tile the 2 × 3 rectangle.

Thus, we have a tricky case only when there exists a set of five tiles, such that four of them are enough to tile the 3 × 2 rectangle, and four of them are enough to tile the 2 × 3 rectangle (see figure 6). We have (up to symmetry) only one set of five tiles satisfying this condition. But, according to the number of glues of strength 2, this set either has final productions which are not rectangles, or does not produce all rectangles. This finishes the proof.

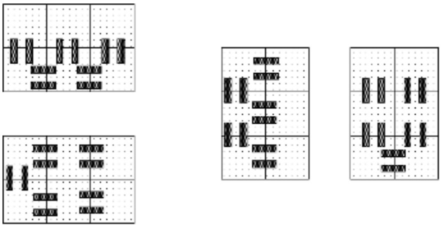


Fig. 6. The possible tilings with four tiles for the 3 × 2 rectangle and the 2 × 3 rectangle, with a seed with two glues of strength 2. A union of two tiling sets (one for each rectangle) contains at least 5 tiles. Moreover, the only set (up to symmetry) of 5 tiles obtained by this way is formed by tiles used in the highest 3 × 2 rectangle and in the rightmost 2 × 3 rectangle.

Proposition 4. *With the tile systems defined above, the parallel time needed to assemble an $m \times n$ rectangle is $m + n$ while the parallel time needed to assemble an $n \times n$ square is $3n - 5$.*

Proof. We give a proof for the squares (rectangles are similar). Notice first that for any shape S with a marked position (x, y), the parallel time to assemble S with the seed at (x, y) is at least $max_{(x',y')\in S}\{d((x, y), (x', y'))\}$, no matter which tileset is used. The parallel time needed to put the tile at $(n - 1, n)$ is at least $2n - 3$ since that is the distance l_1 between $(0, 0)$ and $(n, n - 1)$. This tile is the only one in the line $\{(x, n), 0 \leq x \leq n\}$ with a glue of strength 2 on its south side, thus it has to be put before any other in this line. Thus, the tile at $(n, 0)$ cannot be put before the step $2n - 3 + n - 2 = 3n - 5$. It is then easy to see that this bound is in fact reached, and that the square can be assembled in time $3n - 5$.

4 Diamonds

Notice that a square corresponds to the set $\{(x, y) \mid d_\infty[(x, y), (0, 0)] \leq n\}$, with $d_\infty[(x, y), (x', y')] = \max\{|x - x'|, |y - y'|\}$. If we change the metrics towards the more "natural" $d_1[(x, y), (x', y')] = |x - x'| + |y - y'|$, then the induced shape is the diamond D_n that appears in Figure 7. The problem of producing diamonds is much more complicated than those we tackled before. Any naive approach seem not to work.

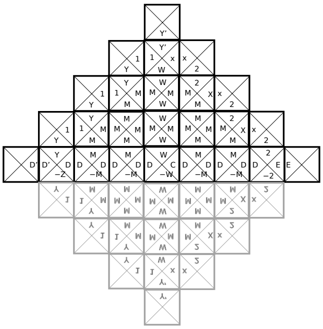
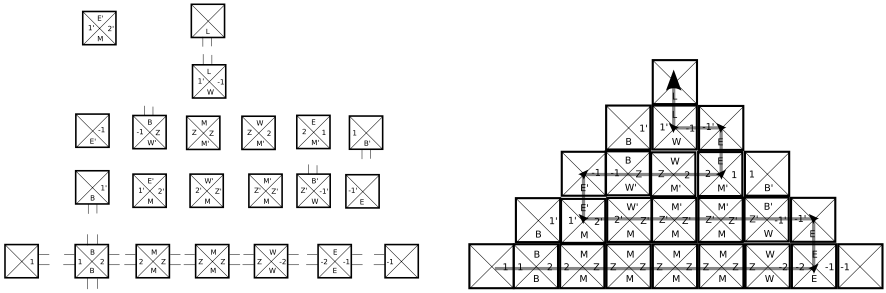

Y'
Y' 1 x x 2 Y W 1
Y 1 1 M W M M X x 2 Y M W M 2 M
Y 1 1 M M M W M M M M M 2 M X x 2 Y M M W M M 2
Y M M W M M 2 D' D' D D D D D D C D D D D D E E −Z −M −M −W −M −M −2

Fig. 7. An n-diamond

A first possible approach is given by the tileset of Figure 8(a). This tileset assembles the upper halves of diamonds. By using this set and its mirror image, one can assemble the set of all diamonds. The tileset works by "knitting" the half-diamond row after row, going back and forth (see Figure 8(b)). The sequentiality of this approach is the cause of the quadratic parallel time $n^2/4$. We are going to show how to lower this bound. More precisely, we are going to construct diamonds in linear parallel time with the help of a very particular and non-trivial cellular automaton called "the firing squad".



(a) The tileset for assembling half-diamonds

(b) The assembly of a half-diamond

Fig. 8. Knitting diamonds

4.1 Simulating the Firing Squad CA by Self-assembly

The firing squad automaton, detailed in [3], is a cellular automaton which, from a line of n cells, such that the first and n-th cells are in special states G_l and G_r, and the other in initial state s, evolves in such a way that they all enter the final state F for the first time at time n, and all other cells remain in a quiescent state ρ.

We will represent, for simplicity, the tiles rotated in 45°. Let us first consider the set of six tiles of Figure 9. The tiles are, from left to right, t_α (the seed), t_{G_l}, t_β, t_s, t_{G_r} and t_γ. The colors $\alpha, \beta, \gamma \notin Q_{FS}$. The color *null* is omitted. For instance, $\sigma_N(t_\alpha) = \sigma_S(t_\alpha) = \sigma_W(t_\alpha) = null$.

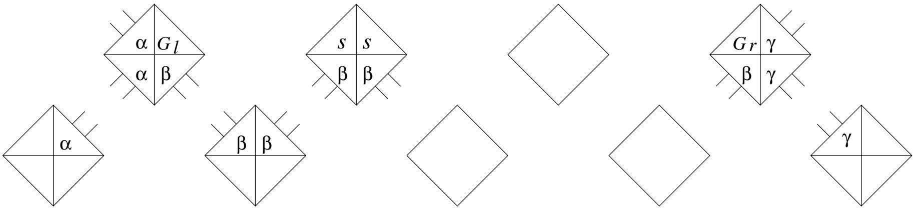


Fig. 9. The set of six tiles that codifies the initial configuration

As it is schematically explained in Figure 10, the assemblying process of these tiles is such that the structure they produce represents the initial configuration $\ldots\rho\rho\rho G_l sss \ldots sss G_r \rho\rho\rho \ldots$.

The size of the initial structure of Figure 10 determines the size of the diamond the tile system is going to produce. This part of the self-assembly process is, in fact, the only nondeterministic one. Therefore, the expected size of the diamond can be calculated as a function of the concentrations of the previously introduced

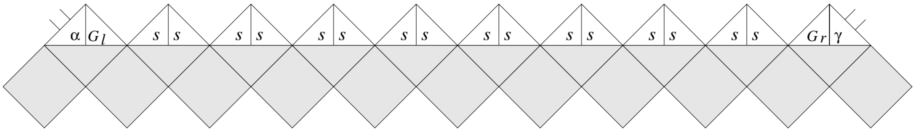


Fig. 10. The way the initial configuration $\ldots \rho\rho\rho G_l sss \ldots sss G_r \rho\rho\rho \ldots$ is assembled

tiles. Moreover, the only concentrations relevant for the process are those of t_s and t_{G_r}. Let $0 < A < 1$. Let us define the concentrations as follows:

$$P(t_s) = A(1 - (n-1)^{-1}),\ P(t_{G_r}) = A(n-1)^{-1}.$$

The only requirement for the concentrations of the other tiles (the four already introduced and those to come) is that they must be positives with their sum being $1 - A$.

There are two other classes of tiles: transmission tiles and transition tiles. The transmission tiles are divided into six subclasses: left-border, internal, right-border, upper-left-border, upper-border, upper-right-border. More precisely, for all $a, b \in Q_{FS} \setminus \{F\}$, the transmission tiles are constructed in Figure 11.

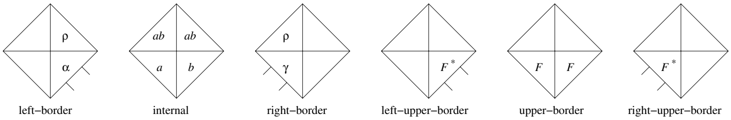


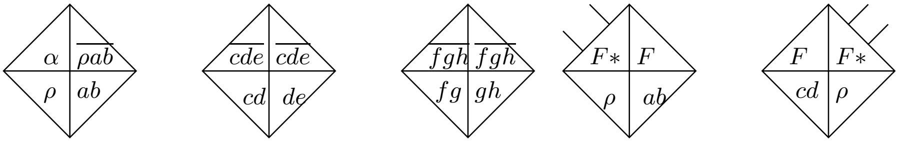
Fig. 11. Transmission tiles

The transition tiles are divided into five subclasses: left-border, internal, right-border, upper-left-border and upper-right-border.

More precisely: let $a, b, c, d, e, f, g \in Q_{FS}$ be such that $\delta_{FS}(\rho, a, b) \neq F$ and $\delta_{FS}(f, g, \rho) \neq F$. The (a, b)-left-border, (c, d, e)-internal and (f, g)-right-border tiles are constructed in Figure 12(a).

Finally, let $a, b, c, d \in Q_{FS}$ be such that $\delta_{FS}(\rho, a, b) = \delta_{FS}(c, d, \rho) = F$. The upper-left-border and upper-right-border tiles are constructed in Figure 12(b).

From the previously defined construction follows the last propositions.



(a) Left-border, internal, right-border.

(b) Upper-left-border, upper-right-border.

Fig. 12. Transition tiles; $\overline{xyz}$ represents $\delta_{FS}(xyz)$

Proposition 5. *The tile system defined above uniquely produces diamonds. If N is the random variable corresponding to the length of the diagonal, then $\mathbb{E}(N) = 2n + 1$; and the parallel time needed by the above tile system to assemble a diamond of size n is $4n - 6$.*

Proof. The parallel assembly follows the simulation of the CA, as shown on Figure 13. By induction one gets that the last tile of the kth row is added in the $2n + k - 4$ parallel transition for $k > 0$. As the assembly is complete when the $(2n - 2)$th row of the simulation is complete, the parallel time for the assembly is $4n - 6$.

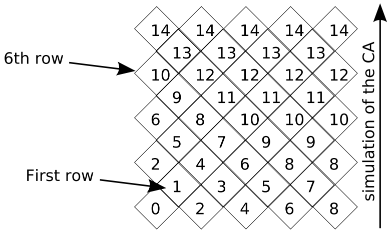


Fig. 13. Parallel assembly of a diamond

References

1. L. Adleman, Q. Cheng, A. Goel and M. Huang. "Running time and program size for self-assembled squares". In Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC 2001), 740-748, 2001.
2. G. Aggarwal, M. H. Goldwasser, M-Y. Kao and R. T. Schweller. "Complexities for generalized models of self-assembly". Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2004), 880–889, 2004.
3. J. Mazoyer and N. Reimen. "A linear speed-up theorem for cellular automata". Theoretical Computer Science 101 (1992), 59-98.
4. P. Rothemund . "Theory and experiments in algorithmic self-assembly". Ph. D. Thesis, University of Southern California, 2001
5. P. Rothemund and E. Winfree. "The program size complextity of self-assembled squares". In Proceedings of the 32nd Annual ACM Symposium on Theory of Computing (STOC 2000), 459-468, 2000.
6. H. Wang. "Proving theorems by pattern recognition". H. Bell System Technical Journal, 40:1-41, 1961.
7. E. Winfree. "Algorithmic self-assembly of DNA". Ph.D. thesis, California Institute of Technology, Pasadena, 1998.

One-Input-Face MPCVP Is Hard for L, But in LogDCFL

Tanmoy Chakraborty[1] and Samir Datta[2]

[1] Dept. Of Computer and Information Science, Univ. of Pennsylvania, USA
tanmoy@seas.upenn.edu
[2] Chennai Mathematical Institute, India
sdatta@cmi.ac.in

Abstract. A monotone planar circuit (MPC) is a Boolean circuit that can be embedded in a plane, and that has only AND and OR gates. Yang showed that the one-input-face monotone planar circuit value problem (MPCVP) is in NC^2, and Limaye et. al. improved the bound to LogCFL. Barrington et. al. showed that evaluating monotone upward stratified circuits, a restricted version of the one-input-face MPCVP, is in $\mathsf{LogDCFL}$. In this paper, we prove that the unrestricted one-input-face MPCVP is also in $\mathsf{LogDCFL}$. We also show this problem to be L-hard under quantifier free projections.

Keywords: L, LogDCFL, monotone planar circuits.

1 Introduction

The problem of evaluating Boolean circuits is a widely studied problem in complexity theory. In [12], the problem of evaluating a Boolean circuit (CVP) was shown to be P-complete under logspace many-one reductions. Special cases of CVP, namely, the *monotone* CVP and the *planar* CVP, have also been shown to be P-complete in [9]. However, a special case of both these versions, the *planar monotone* CVP (MPCVP), is known to be in NC.

It was shown in [10] that upward stratified MPCVP, a special case of MPCVP (see Section 2 for definitions), is in NC^2. The upper bound for this problem was subsequently improved to LogCFL in [7], and quite recently to $\mathsf{LogDCFL}$ in [5].

A less restrictive case, the layered upward MPCVP, was shown to be in NC^3 in [11]. Independently and in parallel, it was shown in [15] and [6] that general MPCVP is in NC^4 and NC^3 respectively.

In [15], it was shown that one-input-face MPCVP, a less restricted case than upward stratified, is in NC^2. Recently, it was shown in [13] that one-input-face MPCVP is in $\mathsf{L}(\mathsf{PDLP} \oplus \mathsf{LogDCFL}) \subseteq \mathsf{LogCFL}$. ($\mathsf{PDLP}$ is the problem of finding the longest path in a planar DAG. Its best known upper and lower bounds are NL and L, respectively.) The upper bound for general MPCVP was also improved to $\mathsf{AC}^1(\mathsf{LogCFL}) = \mathsf{SAC}^2$ in [13].

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 57–68, 2006.

Cylindrical and toroidal circuits were also discussed in [13]. Stratified monotone cylindrical circuits were shown to be in $\mathsf{LogDCFL}$, one-input-face monotone cylindrical circuits in $\mathsf{L}(\mathsf{PDLP} \oplus \mathsf{LogDCFL})$, and monotone cylindrical circuits(in full generality) in $\mathsf{AC}^1(\mathsf{LogDCFL})$. Toroidal circuits were shown to be in SAC^2.

The main result of this paper is that **one-input-face MPCVP is in LogDCFL**. Our method is inspired chiefly from the insights about grid graphs and single source planar graphs developed in [2] and [1]. Our result has been mentioned as personal communication in [13], and we are grateful to its authors for valuable discussion.

We also show that L is a lower bound for one-input-face MPCVP, *i.e.* one-input-face MPCVP is L-hard under quantifier free projections. As corollary to the main result of this paper, we infer that one-input-face cylindrical circuits, can be evaluated in $\mathsf{LogDCFL}$.

The rest of the paper is organized as follows: Section 2 gives some necessary definitions and basic or known facts, Section 3 gives an overview of the proof of our main result, while Section 4 presents its details. In Section 5, we show the L-hardness of one-input-face MPCVP under quantifier free projections, and in Section 6, we summarize the results. Section 7 provides our conclusion.

2 Definitions and Facts

A *Boolean* circuit is a circuit with AND, OR and NOT gates, apart from the input gates. The gates (as vertices) and wires (as edges directed towards the gate for which it is an input wire) of the circuit form a directed acyclic graph (DAG). We shall consider Boolean circuits which also have COPY gates of fan-in one: a COPY gate outputs 1 if and only if its input is 1. Note that the behaviour of a COPY gate is the same as an AND or OR gate with fan-in one.

Circuit value problem (CVP) is the problem of evaluating a circuit when values of the input gates are specified. A circuit is called *monotone* if it does not have any NOT gate. A circuit is called *planar* if its underlying DAG has a planar embedding. MPCVP refers to the restriction of CVP in which the circuit is monotone as well as planar.

A planar circuit is said to be *one-input-face* if it has a planar embedding such that all the input gates are on a single face. The planar embedding need not be given as part of the input, as the following lemma shows.

Lemma 1. *An appropriate planar embedding for a one-input-face circuit can be found in logspace.*

Proof. Consider the planar DAG G corresponding to the circuit C. Add a source vertex s in G, and add edges from s to all the input gates, to obtain a graph G'. Since C is one-input-face, G' is also planar. Find a planar embedding of G', and delete s to get the required embedding for G. A planar embedding can be computed by a logspace transducer, since it was shown to be in $\mathsf{FL}^{\mathsf{SL}}$ in [3], and it was proved that $\mathsf{SL} = \mathsf{L}$ in [14]. □

A planar embedding can be specified by listing the edges incident on each vertex, in cyclic order around the vertex. Such a specification is called a *combinatorial embedding.* A planar embedding is said to be *bimodal* if all the incoming edges at every vertex appear consecutively in the cyclic ordering. For a bimodal planar embedding, we can define the *clockwise-most* and *anticlockwise-most* incoming and outgoing edges at every vertex v without any ambiguity. We can, infact, order all the incoming edges and all the outgoing edges, according to their cyclic ordering, clockwise or anticlockwise.

A planar DAG is called an SSPD if it has a single source (vertex with indegree zero), and a single sink (vertex with outdegree zero). It is well known (*e.g.*, see [1],[15]) that any planar embedding of an SSPD is bimodal. A planar DAG is called an SMPD if it has a single source, but can have multiple sinks.

Similar to planar circuits, one may also consider cylindrical circuits (i.e. embeddable on the surface of a cylinder), and toroidal circuits (i.e. embeddable on the surface of a torus). Please see [13] for definition and properties of such embeddings.

A circuit is said to be *layered* if there is a partition of the vertex set $V = V_0 \cup V_1 \cup V_2 \ldots V_k$, such that all the edges go from V_i to V_{i+1} for some i. Each subset of the partition is called a *layer.* A layered circuit is said to be stratified if there is such a partition, in which all the input gates (vertices) are in V_0. For layered circuits, it is important that the input provides the layering information; all the previous results critically use this fact. Finding a layering for general circuits that can be layered is not known to be in LogDCFL.

A circuit (graph) is said to be *upward planar* if there is a planar embedding in which every edge is monotonically increasing in the upward, or any particular, direction. A circuit (graph) is said to be *upward layered (stratified)* if it is layered (stratified), and the layers give an upward planar embedding. Clearly, an upward stratified circuit is also a one-input-face circuit.

LogCFL and LogDCFL are the classes of languages that are logspace many-one reducible to non-deterministic and deterministic context-free languages, respectively. LogDCFL can be alternately described as the class of languages decidable by a logspace Turing machine that is also provided with a stack, which runs in polynomial time. The following facts are known:

- $\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{LogCFL}$,
- $\mathsf{L} \subseteq \mathsf{LogDCFL} \subseteq \mathsf{LogCFL}$, and
- $\mathsf{LogCFL} = \mathsf{SAC}^1 \subseteq \mathsf{AC}^1 \subseteq \mathsf{NC}^2$.

Grid graphs are planar graphs whose vertices are a subset of the integral points of a finite two-dimensional grid (called *grid points*), and whose edges are either from (i, j) to $(i+b, j)$ (horizontal edge), or from (i, j) to $(i, j+b)$ (vertical edge), where $b \in \{-1, 1\}$. A grid graph has the naturally defined directions up, down, left and right, which are synonymous with north, south, west and east, respectively. We follow the convention that the first coordinate increases rightward, and call it the *rightward/eastward coordinate*, while the second coordinate increases downward, and we call it the *downward/southward coordinate.* [1] and [2] are good references for terminology and facts associated with grid graphs.

A grid graph is said to be *1-forbidden* if it has edges only in three of the four directions. A grid graph is said to be *2-forbidden* or *layered* if it has either rightward or leftward edges, and has either upward or downward edges. Note that a layered grid graph is upward layered (view the grid graph diagonally). Note that a layered grid graph, viewed diagonally, is also an upward layered graph. Each layer consists of all the vertices that lie on a line parallel to the diagonal, and the ordering of the layers can be deduced easily in logspace.

The problem ORD is defined as reachability from a vertex s to another vertex t in a directed graph, consisting of n vertices $v_1, v_2 \dots v_n$ and $(n-1)$ edges (given in the input as ordered pairs of vertices), such that the graph is a directed path. Every vertex v has a unique successor $S(v)$. An equivalent definition of the problem in terms of total orders is given in [8].

It was shown in [8] that ORD is L-complete under quantifier free projections (qfp's). For details on these extremely low level reductions please see [8].

3 Overview

In [13], one-input-face MPCVP was reduced to upward stratified MPCVP, by making oracle calls to the PDLP problem, which finds the longest path in a planar DAG, and then the LogDCFL algorithm given in [5] was used to solve the one-input-face MPCVP in $\mathsf{L}(\mathsf{PDLP} \oplus \mathsf{LogDCFL})$.

We prove that the one-input-face MPCVP is in LogDCFL, by finding a logspace reduction from one-input-face MPCVP to the upward stratified MPCVP. This result would have followed trivially from the algorithm in [13] if PDLP were in LogDCFL, but such a result has not yet been proved, and, for all we know, PDLP can be NL-hard. We take a completely different approach to bypass the PDLP problem and obtain a logspace reduction.

3.1 Graph-Circuit Conversion

In this paper, we shall often store a circuit as a DAG G, with vertices corresponding to gates and edges corresponding to wires. For interpreting G as a circuit, it is required that every vertex carries exactly one of the labels 0, 1, AND, OR, COPY and SRC. The label SRC shall indicate the dummy vertices, that are not present in C. The other labels shall indicate the type of the gate corresponding to the vertex. Further, one of the vertices carries a second label of OUTPUT, which will correspond to the output gate of the circuit. Note that it is possible for a DAG to have a labelling that cannot be interpreted as a meaningful Boolean circuit.

We shall use the following *conversion algorithm*, which, given a DAG G and a labelling of its vertices, decides if the labelling *valid*, *i.e.* whether it can be interpreted as a meaningful circuit, and also produces the unique circuit corresponding to G, if it is meaningful:

1. If some vertex labelled COPY does not have indegree one, report that the labelling is not valid.

2. Delete all vertices (and edges incident on them) that should not be there in the circuit. These include vertices labelled SRC, and also those vertices v labelled COPY, such that there is a path from another vertex u, labelled SRC, all whose internal vertices are labelled COPY.
3. Replace the remaining vertices by gates according to the labelling, and the edges by wires. The gate corresponding to the vertex labelled OUTPUT is marked as the output gate of the circuit produced.

Since the hardest step in the conversion algorithm involves checking reachability in graphs by *simple* paths (paths whose internal vertices have total degree 2 in the graph), the algorithm can be implemented in logspace.

We shall refer to the circuit obtained by the conversion algorithm as the circuit corresponding to the graph. For any vertex that is not deleted by the algorithm, the gate corresponding to it will have a value in the evaluation of the circuit, which we shall refer to as the *value at the vertex.*

Note that, given a circuit C, it is trivial to construct a graph G, such that the conversion algorithm applied on G yields C.

3.2 Steps of the Reduction

Given a one-input-face MPC C, consider its underlying single-source planar DAG, with vertices labelled accordingly. We add a source vertex s to the graph, with edges to all the vertices labelled 0 or 1, and label it as SRC. Let this graph, which is an SMPD, be G.

The reduction then proceeds sequentially in 5 major steps. Each step takes the output of the previous step as its input, and uses it to produce some output, in logspace. Step 1 takes G(with its labelling) as input. Each of steps 1-4 output a planar DAG (that has certain useful properties) with a valid labelling. We shall ensure that the value of the circuit corresponding to the output of each step is the same as that of the input circuit C. Step 5 produces an upward stratified circuit, hence completing the reduction.

The chief properties of the output of the each step is listed below:

1. An SSPD G_1.
2. An SSPD G_2 whose total degree at each vertex is bounded by 3, and the indegree and outdegree by 2.
3. A 1-forbidden grid graph G_4, that is also an SMPD.
4. A layered grid graph G_5, that is also an SMPD.
5. An upward stratified circuit C''.

The upward stratified circuit C'' obtained at the end of step 5 can then be evaluated in LogDCFL, as described in [5].

4 Details of the Reduction

In this section, we provide the necessary details about how to implement the steps, outlined in the overview, in logspace, and also show that the circuit value is preserved.

4.1 Step 1

Suppose that a vertex u does not have a path to t, the vertex labelled OUTPUT. Then the value at t is independent of the value at u. So, deleting u does not affect the circuit value. If we delete all such vertices, then it is easy to see that the resulting graph has a single source s and a single sink t, *i.e.* the resulting graph G_1 is an SSPD, and the circuit value remains unchanged. It was shown in [1] that reachability in single-source planar DAGs is in L, so G_1 can be constructed from G by a logspace transducer.

4.2 Step 2

We compute a planar embedding (combinatorial) of G_1. This can be done in logspace, by Lemma 1. Note that since G_1 is an SSPD, the embedding is bimodal.

To reduce the degrees of the vertices as required, we replace each vertex v of G_1 by a gadget, to obtain the graph G_2. It comprises two directed binary trees, one with its root as its source, and the other with its root as its sink. We shall refer to the former as *outgoing tree*, and to the latter as *incoming tree*. There is also an edge from the root of the incoming tree to that of the outgoing tree. Both the trees have depth at most $\lceil \log |V| \rceil$, where $|V|$ is the number of vertices in G_1. The number of leaves in the incoming tree (*incoming leaves*) is equal to the indegree of v, and the number of leaves in the outgoing tree (*outgoing leaves*) is equal to the outdegree of v. All the vertices of the outgoing tree are labelled COPY. If v were labelled AND, OR or SRC, all the vertices of the incoming tree are labelled AND, OR or SRC, respectively. If v were labelled 0 or 1, then the vertices of the incoming tree, except its root, are labelled COPY, while its root is labelled 0 or 1. Note that the gadget corresponding to s will not have an incoming tree, and the gadget corresponding to t will not have an outgoing tree. For s, the root of the outgoing tree is labelled SRC, and for t, the root of the incoming tree is labelled OUTPUT.

Note that the incoming leaves and the outgoing leaves are arranged in a bimodal fashion, *i.e.* the incoming leaves appear consecutively in a cyclic ordering. Now, for every edge $e = (u, v)$ in G_1, which is the i^{th} outgoing edge of u and the j^{th} incoming edge of v(unambiguously defined, due to bimodality in G_1), we put an edge in G_2 from the i^{th} outgoing leaf of the gadget for u to the j^{th} incoming leaf of v. Because of bimodality in G_1, G_2 is planar. Also, G_2 is an SSPD, satisfying the degree constraints. It is easy to see that the value of the circuit for G_2 is the same as that for G_1.

Since the gadget for each vertex is dependent only on its indegree and outdegree, they can be constructed by a logspace transducer. The other edges of G_2 can also be added by the same transducer.

4.3 Step 3

This is the most involved step in our reduction. We first convert G_2 into an SMPD G_3 with certain advantageous features, that has the same circuit value as G_2, and then embed G_3 as a *1-forbidden* grid graph G_4, by only subdividing some of the edges (*i.e.* replacing edges by simple paths) of G_3. We shall label the new vertices created due to the subdividing as COPY, and it is easy to see that the circuit value will remain unchanged. Note that the degree constraints achieved in Step 2 are also not violated.

The process of embedding in the grid is similar in spirit to the process given in [2], where it was shown how to embed a planar graph in a grid using only logspace, preserving reachability. Here, we have an SSPD to embed instead of a general planar graph, while we additionally require that the grid graph produced should be monotone along one axis (we shall ensure that G_4 has no westward edge), and also want to preserve circuit value. This is significant, because reachability is precisely evaluation of circuits with only OR gates, and hence possibly easier to preserve than values of circuits with both AND and OR gates.

Using the mentioned embedding of G_2, we construct a subgraph H, by deleting all incoming edges except the clockwise-most one at every vertex of G_2 except the source and sink (the clockwise-most edge is unambiguously defined, due to bimodality). Delete all but one (arbitrarily chosen) of the edges incoming to the sink t. It is easy to see that H is a directed tree spanning all vertices, with s as its root.

We can now classify the edges of G_2 as tree edges (those present in H) and non-tree edges. The non-tree edges can be further classified as *forward edges* (from a vertex to its descendant in H), and *cross edges* (between different subtrees). Since G_2 is a DAG, there is no *back* edge (from a vertex to its ancestor). Due to the bounded degree of G_2, H is a binary tree. We perform an Euler traversal (same as a *dfs* traversal for a tree) of H starting at s, choosing the anticlockwise-most unexplored edge at every stage (we consider the embedding of H derived from G_2). In the beginning, at s, we make an arbitrary choice of the edge to explore first. We write down the *discovery time* $d[v]$ and the *finishing time* $f[v]$ of every vertex v using a logspace transducer.

Before describing the reduction any further, we need the following lemmas whose proofs we omit.

Lemma 2. *Suppose H is drawn as the dfs-tree, mentioned above, in standard fashion, with the child explored first drawn as the left child at every vertex. The combinatorial embedding of H thus obtained is the same as that derived from G_2.*

Hence, it is possible to add and embed the non-tree edges to the dfs-tree in a planar way such that the combinatorial embedding is the same as that of G_2 at the end of the previous step. The dfs-tree helps us define the left and right of every vertex that is not the source or a leaf in H. There cannot be any non-tree edge incoming to or outgoing from a vertex u between its left and right child, due to bimodality and degree constraint, respectively. Hence, every non-tree edge is incoming to and outgoing from every vertex from either its left or its right. For leaves, there is no distinction between left and right, and we shall take the liberty of either.

Lemma 3. *Any non-tree edge (u, v), is incoming to v from the left of v.*

For any two vertices u and v that do not share an ancestor-descendant relationship, we say that u is to the left of v if the discovery and finishing times of u is less than that of v, and vice versa otherwise. We say that a cross edge (u, v) is *leftward* or *rightward*, depending on whether u is to the right or left of v, respectively. We say that a forward edge (u, v) is leftward or rightward, depending on whether the edge is outgoing from the right or left of u, respectively.

Notice that Lemma 3 does not imply that there are no leftward edges, since its quite possible that the origin u of an edge (u, v) is to the left of the terminal v. It just says that even such edges approach v from the left (see Figure 1).

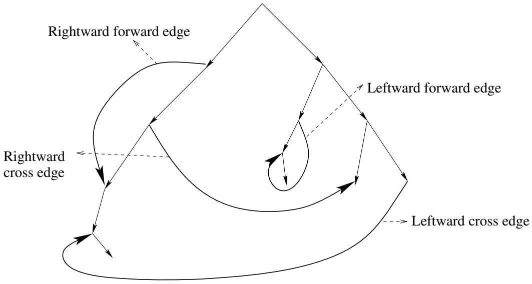


Fig. 1. Possible types of non-tree edges

If we neglect the direction of the edges, every non-tree edge (u, v) added to H produces a unique cycle, consisting of the undirected tree-path between u and v, and the edge (u, v) itself. We call the curve formed by the edges of this cycle as the *characteristic closed curve* of (u, v).

Lemma 4. *Suppose* (u, v) *is a rightward non-tree edge. Then* t *cannot be strictly inside characteristic closed curve of* (u, v)*.*

Lemma 5. *Suppose* (u, v) *is a leftward non-tree edge. Then* t *cannot be strictly outside the characteristic closed curve of* (u, v)*.*

We shall now construct a graph G_3 with an analogous tree H', such that there is no leftward non-tree edge, and the circuit value remains unchanged. Suppose there are k leftward non-tree edges. To construct G_3, we make $k+1$ disjoint copies of G_2 without the leftward edges, and label the copies $1, 2 \dots k, k+1$. For every leftward edge (u, v) in G_2 and $\forall i,\ 1 \leq i \leq k$, add an edge between u of the i^{th} copy and v of the $(i+1)^{th}$ copy of G_2. Finally, we add a new source s', and add edges from s' to all the $k+1$ copies of s. Expectedly, we label s' as SRC. H' consists of the copies of H, plus s' and its outgoing edges.

We claim that G_3 is planar. To show this, we observe that from lemma 5 and planarity, it follows that the leftward edges in G_2 are *nested*, *i.e.* if e_1 and e_2 are two leftward edges, either E_1 is contained in the characteristic closed curve of e_2, or vice versa. Thus the cross edges between the copies do not intersect, and, infact, those between any two consecutive copies are also nested. Further, no such edge is nested within a rightward edge, for that would contradict lemma 4. So, these edges between copies do not intersect any other edge.

Note that G_3 is no longer an SSPD, but an SMPD, hence there is no clear choice for the output gate (vertex) of the circuit (graph). For every vertex v of G_2, we say that the i^{th} copy of v in G_3 gets the *correct* value if its circuit value in G_3 is the same as that of v in G_2, otherwise we say that it gets the *wrong* value. We claim that the $(k+1)^{th}$ copy of t has the correct value, and hence we shall label it as the output gate in G_3.

Suppose the k leftward edges in G_2 are $(u_1, v_1), (u_2, v_2) \dots (u_k, v_k)$, with (u_2, v_2) nested inside (u_1, v_1), (u_3, v_3) nested inside (u_2, v_2), and so on, (u_k, v_k) being the innermost leftward non-tree edge. Note that, due to degree constraints, $v_1, v_2 \dots v_k$ are all distinct vertices. To prove our claim, we shall use the following lemmas:

Lemma 6. *A vertex in G_3, that is not in the first copy of G_2, can get the wrong value only if at least one of the vertices, whose values are fed into it, get the wrong value.*

Lemma 7. *There is no path from v_i to u_j or from v_i to v_j, if $i \neq j$, in G_2, $\forall 1 \leq j \leq i \leq k$.*

We say that the i^{th} copy of a vertex v has *primitive error* if it gets the wrong value, but all the vertices in the i^{th} copy, that have an edge to it in G_3, get the correct value.

Lemma 8. *1. If the i^{th} copy of a vertex v gets the wrong value, it must be reachable from a vertex of the i^{th} copy that has primitive error.*

2. Also, no vertex, other than $v_1, v_2 \dots v_k$, can have primitive error in any of its copy.

3. A vertex v_j can have a primitive error in the i^{th} copy only if u_j gets a wrong value in the $(i-1)^{th}$ copy.

We shall prove the following statement using induction:

Lemma 9. *In the i^{th} copy, u_j and v_j get the correct value, and hence do not have a primitive error, $\forall 0 < j < i \leq (k+1)$.*

Putting $i = k+1$ in the Lemma 9, we get that the $(k+1)^{th}$ copy does not have any vertex with primitive error, and hence, by lemma 8 (1), no vertex in the $(k+1)^{th}$ gets the wrong value. Hence our claim is proved.

Note that our construction has ensured that G_3 consists of the tree edges of H' and rightward non-tree edges only. Also note that a rightward non-tree edge can start from the left of a vertex, go leftwards, and then go rightwards to end at a vertex to the right of its starting vertex. But, for embedding in a grid in a 1-forbidden fashion, we demand that every cross edge should always be rightwards in direction. In precise terms, we demand the following:

- Every non-tree edge should be a cross edge.
- Every such cross-edge should be rightward.

– Every such cross-edge should start from the right of a vertex and ends at the left of a vertex.

By Lemma 3, our construction has ensured that every non-tree edge ends at the left of its end-point. For every non-tree edge (u, v) that starts from the left of u, we divide (u, v) into two edges, (u, w) and (w, v), and add (u, w) to H', so that w becomes the left child of u. The non-tree edge (w, v) starts from a leaf, and so trivially satisfies the condition. Clearly, the degree constraints are not violated. Since the forward edges present in G_3 must be rightward, and hence start from the left of a vertex, this process gets rid of all forward edges.

For simplicity, we continue to call the modified graph as G_3, and the tree as H'. The new vertices generated due to the subdivisions are labelled as COPY, clearly the circuit value remains unchanged.

To complete the step, we now embed G_3 in a grid, by only subdividing its edges. The vertices formed due to subdividing are labelled COPY, and, clearly, the circuit value is preserved. This part of the step is almost identical to the process of embedding any planar graph in a grid, given in [2].

In the embedding, each edge of G_3 corresponds to a grid path in the grid graph G_4 thus produced, and every vertex of G_3 correspond to a grid point in G_4. If we view these grid paths as the curved edges of G_3 drawn on the plane, the embedding process ensures that the combinatorial embedding of G_3 remains unchanged. This fact, coupled with the carefully chosen parameters in the process, ensure that no two grid paths, that represent two edges of G_3, intersect. The nature of the non-tree edges of G_3 ensures that G_4 is 1-forbidden.

4.4 Step 4

Barrington ([4]) gave a logspace conversion from a 1-forbidden grid graph to a layered grid graph, preserving reachability. We observe that the reduction, with an easy-to-compute labelling, preserves circuit value as well.

4.5 Step 5

We apply the conversion algorithm on G_5 to obtain a circuit C'. Since G_5 is a layered grid graph, C' is upward layered (since G_5 had only northeast and southeast edges, each layer consists of the vertices on a particular north-south grid line). Moreover, since G_5 is an SMPD, C' is a one-input-face circuit, with the inputs appearing on the external face.

We convert C' into an upward stratified circuit C''(thus completing the reduction), as follows: For each input gate that is on a layer V_i for some $i > 0$, add a copy of it to V_0, label the original gate as COPY, introduce a COPY gate at all intermediate layers $V_1, V_2 \dots V_{i-1}$, and connect the new gate to the original gate through all these new gates. Again, this operation can be performed in logspace. Since the entire reduction consisted of a constant number of steps, each of which is in logspace, so the entire reduction is in logspace.

5 L-Hardness

In this section, we show that one-input-face MPCVP is L-hard under qfp's, by reducing ORD to it via qfp's.

Given an instance of ORD, we map it to the following instance of one-input-face MPCVP: there is an OR gate for every vertex v_i, which takes as input the gate corresponding to the vertex $S(v_i)$ and a constant gate (the single vertex that has no successor has only a constant gate as input). The constant input is 1 for t and 0 for all other vertices. The gate corresponding to s is made the output gate. Notice that the circuit thus constructed outputs 1 if and only if s precedes t in the ordering induced by S, *i.e.* the problem instance belongs to ORD. Clearly, the circuit is planar, and all constant gates(inputs) appear on the external face.

6 Results

Hence, we have proved that

Theorem 1. *One-input-face MPCVP is in* LogDCFL*, but is* L*-hard under quantifier free projections.*

It was shown in [13] that monotone stratified cylindrical circuits can be evaluated in LogDCFL, by reducing it to monotone upward stratified circuits in logspace, and then using the algorithm given in [5]. It was also shown in [13] that monotone one-input-face cylindrical circuits are in L(PDLP $\oplus$ LogDCFL), by reducing it to monotone stratified cylindrical circuits using oracle calls to PDLP. Since one-input-face cylindrical circuits also have a one-input-face planar embedding (see [13]), so Theorem 1 trivially implies that both these problems are in LogDCFL.

Corollary 1. *One-input-face monotone cylindrical circuits (and hence monotone stratified cylindrical circuits) can be evaluated in* LogDCFL.

7 Conclusion

A close inspection of the logspace reduction, that we have described in Sections 3 and 4, reveals that it does not use the fact that the circuit is monotone, not even the fact that the circuit is Boolean. In other words, given any one-input-face planar circuit (need not be Boolean, *i.e.* gates and wires can take more than two values) with any kind of gates, we can produce an equivalent upward stratified circuit in logspace, provided we are allowed to use COPY gates. Hence, the reduction in this paper can be applied to much more general situations.

The exact complexity of one-input-face MPCVP remains open. In other words, is the problem solvable in L? Or is it hard for LogDCFL? General MPCVP has a larger gap between its lower and upper bounds. It is known to be hard only for L, while the best known upper bound is LogCFL.

Acknowledgement

We thank Nutan Limaye and Meena Mahajan for valuable discussions on the subject. We also thank Dave Barrington and Eric Allender for their comments on a preliminary version of this paper.

References

1. E. Allender, D. A. M. Barrington, T. Chakraborty, S. Datta and S. Roy. Grid Graph Reachability Problems. In *Electronic Colloquium on Computational Complexity, Technical Report TR05-149*, 2005.
2. E. Allender, S. Datta and S. Roy. The Directed Planar Reachability Problem. In *Proc. 25th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), LNCS vol. 3821*, pages 238–249, 2005.
3. E. Allender and M. Mahajan. The Complexity of Planarity Testing. In *Information and Computation, vol. 189(1)*, pages 117-134, 2004.
4. D. A. M. Barrington. Grid Graph Reachability Problems. In *Talk presented at Dagstuhl Seminar on Complexity of Boolean Functions, Seminar number 02121*, 2002.
5. D. A. M. Barrington, C.-J. Lu, P. B. Miltersen and S. Skyum. Searching Constant Width Mazes Captures the AC^0 Hierarchy. In *Proceedings of 15th International Symposium on Theoretical Aspects of Computer Science (STACS), LNCS vol. 1373*, pages 73–83, 1998.
6. A. L. Delcher and S. R. Kosaraju. An NC Algorithm for Evaluating Monotone Planar Circuits. In *SIAM Journal of Computing, vol. 24(2)*, pages 369-375, 1995.
7. P. W. Dymond and S. A. Cook. Complexity Theory of Parallel Time and Hardware. In *Information and Computation, vol. 80(3)*, pages 205–226, 1989.
8. K. Etessami. Counting quantifiers, successor relations, and logarithmic space. In *Journal of Computer and System Sciences , vol. 54(3)*, page 400-411, 1997.
9. L. M. Goldschlager. The Monotone and Planar Circuit Value Problems are Logspace Complete for P. In *SIGACT News, vol. 9(2)*, pages 25–29, 1977.
10. L. M. Goldschlager. A Space-Efficient Algorithm for the Monotone Planar Circuit Value Problem. In *Information Processing Letters, vol. 10(1)*, pages 25–27, 1980.
11. S. R. Kosaraju. On the Parallel Evaluation of Classes of Circuits. In *Proc. 10th annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), LNCS vol. 472*, pages 232–237, 1990.
12. R. E. Ladner. The Circuit Value Problem is Logspace Complete for P. In *SIGACT News*, pages 18-29, 1975.
13. N. Limaye, M. Mahajan and J. Sarma M. N. Evaluating monotone circuits on cylinders, planes, and tori. In *Electronic Colloquium on Computational Complexity, Technical Report TR06-009*, 2006.
14. O. Reingold. Undirected st-Connectivity in Log-Space. In *Proceedings of 37th ACM Symposium on Theory of Computing (STOC), IEEE Computer Society Press*, pages 376–385, 2005.
15. H. Yang. An NC Algorithm for the General Planar Monotone Circuit Value Problem. In *SPDP: 3rd IEEE Symposium on Parallel and Distributed Processing.* ACM Special Interest Group on Computer Architecture (SIGARCH), and IEEE Computer Society, 1991.

Hardness of Approximation Results for the Problem of Finding the Stopping Distance in Tanner Graphs

K. Murali Krishnan and L. Sunil Chandran

Department of Computer Science and Automation
Indian Institute of Science, Bangalore 560012, India
{kmurali, sunil}@csa.iisc.ernet.in

Abstract. Tanner Graph representation of linear block codes is widely used by iterative decoding algorithms for recovering data transmitted across a noisy communication channel from errors and erasures introduced by the channel. The stopping distance of a Tanner graph T for a binary linear block code C determines the number of erasures correctable using iterative decoding on the Tanner graph T when data is transmitted across a binary erasure channel using the code C. We show that the problem of finding the stopping distance of a Tanner graph is hard to approximate within any positive constant approximation ratio in polynomial time unless $P = NP$. It is also shown as a consequence that there can be no approximation algorithm for the problem achieving an approximation ratio of $2^{(\log n)^{1-\epsilon}}$ for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{poly(\log n)})$.

1 Introduction

Linear block codes are widely used for reliable transmission of information across a noisy communication channel. A linear block code may be represented (not necessarily uniquely) as a bipartite graph called a Tanner graph [2]. Interest in such graphical representation is primarily due to the fact that the well known *Iterative Decoding Algorithm* [11] typically operates on a Tanner graph representation of the code. The error correcting capability of a code under iterative decoding depends on the Tanner graph representation of the code used at the receiver-end of the communication channel for decoding. In particular, the *stopping distance* of a Tanner graph T for a binary linear block code C determines the number of *erasures* that the iterative decoding algorithm can correct when C is used for communication across a *binary erasure channel* and decoding is performed by running the iterative decoding algorithm on the graph T [1].

In this paper, we study the computational complexity of the problem of finding the stopping distance of a given Tanner graph. The problem was shown to be NP-hard in [12]. We show that unless $P = NP$ there exists no polynomial time approximation algorithm for the problem achieving any constant approximation ratio. It is also shown that there can be no approximation algorithm for the problem achieving an approximation ratio of $2^{(\log n)^{1-\epsilon}}$ for any $\epsilon > 0$ unless $NP \subseteq DTIME(n^{poly(\log n)})$.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 69–80, 2006.

2 Background

The following sub-section reviews the necessary background in coding theory required for an understanding of the problem and set up the notation for the rest of the paper.

2.1 Codes, Tanner Graphs and Stopping Distance

An (n, k) binary linear block code C (hereafter referred to simply as a code) is a k dimensional subspace of the vector space F_2^n with $0 \leq k \leq n$, where F_2 refers to the binary field. A vector $\mathbf{x} \in F_2^n$ satisfying $\mathbf{x} \in C$ is called a *codeword*. A *generator matrix* G for a code C is a $k \times n$ matrix over F_2 whose rows generate the subspace C. The dual space for a code C called $C^\perp$ is the orthogonal complement of C in F_2^n i.e., $C^\perp = \{x \in F_2^n : x.y = 0\}$, where "." represents the standard dot product in F_2^n. It is well known that that $C^\perp$ is a subspace of F_2^n of dimension $n - k$ [13, p. 244]. An $(n - k) \times n$ generator matrix H for $C^\perp$ is called a *parity check matrix* for C. Note that For any $\mathbf{x} \in C$, $\sum_{i:h_{ji}=1} x_i = 0$ for any $1 \leq j \leq n - k$ with addition carried out over F_2.

Given the parity check matrix $H = [h_{ij}] \in F_2^{(n-k)\times n}$, $0 \leq k \leq n$ for an (n, k) binary linear code, the *Tanner graph* defined by H is the undirected bipartite graph $G = (L, R, E)$ where $L = \{x_i, 1 \leq i \leq n\}$, $R = \{c_j, 1 \leq j \leq n - k\}$ and $E = \{(x_i, c_j) : h_{ji} = 1, 1 \leq i \leq n, 1 \leq j \leq n - k\}$. The set L is called the set of *code symbols* and the set R called the set of *parity checks*. We refer to the set L and R as the set of left and right vertices of G respectively. Figure 1 shows the parity check matrix for a $(3, 1)$ binary linear code and the corresponding Tanner graph.

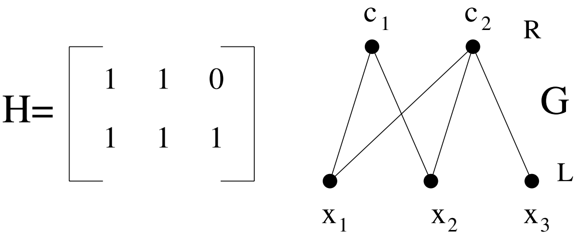


Fig. 1. Parity check matrix and Tanner graph for a (3,1) code

For $S \subseteq L \cup R$, we define $N(S) = \{v : (u, v) \in E, u \in S\}$. $S \subseteq L$ is a *stopping set* if $S \neq \emptyset$ and for all $c_j \in N(S)$, $|N(\{c_j\}) \cap S| \geq 2$ i.e., every vertex connected to some vertex in a stopping set must have at least two neighbours in the stopping set.

The *stopping distance* of a Tanner graph is the size of a stopping set of minimum cardinality in the graph. We define the optimization problem $MINSTOP$ as the problem of determining the stopping distance of a given Tanner graph. The objective of this paper is to study the computational complexity of $MINSTOP$.

The following subsection, though not necessary for reading the rest of the paper, explains the motivation for studying the problem.

2.2 Erasure Channel and Iterative Decoding

A *binary erasure channel* with parameter p called $BEC(p)$ with $p \in [0,1]$ is a stochastic process taking input symbol $x \in \{0,1\}$, producing output symbol $y \in \{0,1,e\}$ with the transition probabilities $Pr(y=0|x=0) = Pr(y=1|x=1) = 1-p$ and $Pr(y=e|x=0) = Pr(y=e|x=1) = p$. The symbol e is called the *erasure* symbol and models the situation where the value of a bit transmitted is unrecognizable at the receiver end. The action of an erasure channel on a vector $\mathbf{x} \in F_2^n$ is defined as the stochastically independent action of $BEC(p)$ on its n component symbols x_i, $1 \leq i \leq n$. The binary erasure channel is a useful model for several practical communication systems.

The problem of (*maximum likelihood*) *decoding* a code C on a channel $BEC(p)$ is that of finding an $\mathbf{x} \in C$ for a given $\mathbf{y} \in \{0,1,e\}^n$ such that $Pr(\mathbf{y}|\mathbf{x}) \geq Pr(\mathbf{y}|\mathbf{x}')$ for all $\mathbf{x}' \in C$. The problem can be shown to be equivalent to finding an $\mathbf{x} \in C$ that agrees with $\mathbf{y}$ in maximum number of positions.

The *iterative decoding algorithm* takes as input a Tanner graph G for an (n,k) code C and a vector $\mathbf{y} \in \{0,1,e\}^n$ and does the following. The algorithm first associates the symbol y_i with the vertex $x_i \in L$ for each $1 \leq i \leq n$. The algorithm then finds a vertex $c_j \in R$ such that only one of the neighbours of c_j — say x_t has value e. Since for any $\mathbf{x} \in C$ the equation $\sum_{x_i \in N(\{c_j\})} x_i = 0$ must be satisfied, the algorithm can solve for the single unknown value x_t so that the above equation holds. The value of x_t so found replaces the old value e in the next iteration. The process continues until either all erasures are solved successfully or every $c_j \in R$ with a neighbour with value e has at least one more neighbour with value e. In the latter case, the algorithm fails to progress and announces a *decoding failure.* The iterative decoding algorithm though sub-optimal is widely used owing to its low computational complexity.

It is not hard to see [1] that decoding failure occurs if and only if the set of vertices in G corresponding to erasures in the input vector $\mathbf{y}$ contains a stopping set as a subset. Hence the stopping distance of a Tanner graph is of interest as it is the minimum number of erasures that can cause a decoding failure when iterative decoding algorithm is used for decoding on the graph. Considerable analysis on the structure of stopping sets in the Tanner graphs of various code families as well as design of codes with large stopping distance has been carried out — for example see [3,4,5,6,9,7,10].

The problem of finding the stopping distance of a Tanner graph was shown to be NP-hard in [12]. In this paper, non-approximability results for the problem are derived. The following subsection gives a brief outline of the rest of the paper. The definitions provided are not the most general, but are sufficient for the purposes for this paper.

2.3 Non-approximability

For any $\epsilon > 0$, A $(1+\epsilon)$-approximation algorithm Π for a *minimization problem* P is one that on any instance x of P returns a feasible solution y such that $COST(y) \leq (1+\epsilon)OPT(x)$, where $OPT(x)$ denotes the cost of an optimal solution for the instance x and $COST(y)$ denotes the cost of y with respect to the cost function under consideration. We require that Π runs in time polynomial in the size of the input x. Approximation algorithms have been intensively studied in computer science literature — see for example [14,15,16].

A vertex cover in an (undirected) graph G is a subset S of the vertices of G such that every edge in G has at least one endpoint in the set S. The problem of finding a minimum vertex cover in a graph called $MINVC$ is NP-hard [8, p. 190]. It is known that there exists an $\epsilon_0 > 0$ such that no polynomial time $(1+\epsilon_0)$-approximation algorithm exists for $MINVC$ unless $P = NP$ even for graphs whose vertex degree is bounded by three [15, p. 369]. We shall call the specialization of $MINVC$ to graphs of degree at most 3 as $MINVC(3)$.

Our objective here is to show that there does not exist any polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ for *any* constant $\epsilon > 0$ unless $P = NP$. We achieve this by showing first that if there is a polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$, then using the algorithm we can deduce a polynomial time $(1+4\epsilon)$-approximation algorithm for $MINVC(3)$. As the next step we show that $MINSTOP$ has the interesting property that the existence of a polynomial time $(1+\epsilon)$-approximation algorithm for *some* constant $\epsilon > 0$ implies existence of a polynomial time $(1+\epsilon)$-approximation algorithm for *every* constant $\epsilon > 0$. As a consequence, existence of a polynomial time $(1+\epsilon)$-approximation for $MINSTOP$ for any constant $\epsilon > 0$ would imply existence of a polynomial time $(1+\epsilon)$-approximation algorithm for $MINVC(3)$ for *every* constant $\epsilon > 0$ which is impossible unless $P = NP$.

3 Hardness of *MINSTOP*

This section proves the results claimed in the paper. The following subsection shows that existence of a $(1+\epsilon)$- approximation algorithm for $MINSTOP$ implies existence of a $(1+4\epsilon)$-approximation algorithm for $MINVC(3)$. We follow the standard terminology and notation for representing graphs.

3.1 Reduction from MINVC(3) to MINSTOP

Given an instance of the $MINVC(3)$ problem, namely, an undirected graph $G = (V, E)$, with degree of each vertex bounded by 3. Let $|V| = n$ and $|E| = m$. We assume that G is connected. It is not hard to see that assuming that G is connected does not affect the hardness of approximation results for $MINVC(3)$. Let $V = \{v_1, v_2, ..., v_n\}$ and $E = \{e_1, e_2, ..., e_m\}$. Without loss of generality, we may assume $m > 0$, $n > 0$ and $e_1 = (v_1, v_2)$. We construct undirected bipartite graph $G_1 = (L, R, E')$, where $L = L_0 \cup L_1$ is the left vertex set of G_1, $R = R_0 \cup R_1$ is the right vertex set of G_1, $R_0 = \{e_1, ...e_m\}$, $L_1 = \{e'_1, ...e'_m\}$, $L_0 = \{v_1, ..., v_n\}$ and $R_1 = \{z_1, ..., z_m\}$. The edges of G_1 are connected as follows:

- For each edge $e = (u, v)$ in G. connect $e \in R_0$ to u and v in L_0. In other words, L_0 and R_0 are connected according to the vertex-edge incidence in G.
- Connect $e_i \in R_0$ to $e'_i \in L_1$ for $1 \leq i \leq m$.
- Connect $z_i \in R_1$ to e'_i and e'_{i+1} in L_1, $1 \leq i \leq m - 1$.
- Connect $z_m \in R_1$ to $v_1 \in L_0$ and $e'_1 \in L_1$. (Recall our assumption that $e_1 = (v_1, v_2)$).

The example in Figure 2 illustrates the construction. We construct a graph G_2 which is identical to G_1 except that the edge (z_3, v_1) between R_1 and L_0 is *replaced* with the edge (z_3, v_2).

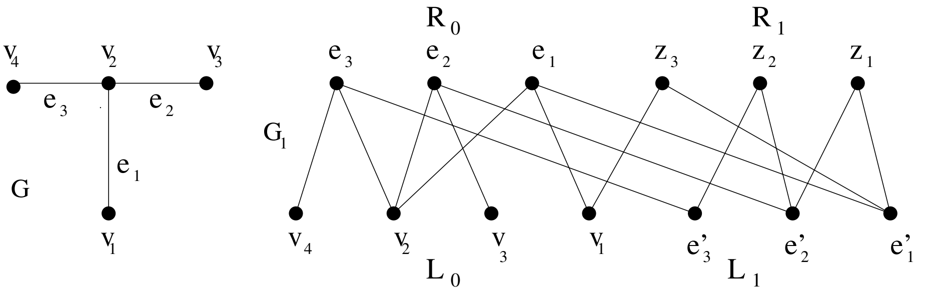


Fig. 2. Construction of the graph G from G

Given any subset S of vertices in G, we can identify the set S as a subset of L_0 in G_1 and G_2. A similar identification works in the reverse direction as well. We will be using this identification repeatedly without particular mention of that fact to avoid cumbersome notation. We first establish some observations about stopping sets in G_1 and G_2. The first observation is an immediate consequence of the way edges are connected between R_1 and L_1 and shows that any stopping set S' in G_1 (or G_2) containing at least one vertex in L_1 must contain the whole of L_1.

Lemma 1. *Let S' be a stopping set in G_1 (or G_2). Suppose $S' \cap L_1 \neq \emptyset$ then $L_1 \subseteq S'$.*

Proof. Let $e'_i \in L_1$ be contained in S'. If $i < m$, z_i which is a neighbour of e'_i must have two neighbours in S' (for S' to satisfy the stopping set condition). Since the only neighbours of z_i are e'_i and e'_{i+1} we have $e'_{i+1} \in S'$. Similarly, for $i > 1$, z_{i-1} must have two neighbours in S' and hence $e'_{i-1} \in S'$. Extending the argument, we see that $e'_i \in S'$ for some $1 \leq i \leq m$ if and only if $e'_i \in S'$ for every $1 \leq i \leq m$ proving the lemma.

The following lemma proves that any stopping set S' in G_1 (or G_2) must contain the whole of L_1. This is a consequence of the fact that G is connected.

Lemma 2. *Let S' be a stopping set in G_1 (or G_2). Then $L_1 \subseteq S'$.*

Proof. By Lemma 1, it is enough to prove that there exists some $e'_i \in L_1$, $1 \leq i \leq m$ satisfying $e'_i \in S'$. Assume on the contrary that $S' \cap L_1 = \emptyset$. Then we first show that $L_0 \subseteq S'$. If not, then there must exists an edge $e_i = (v_j, v_k)$, $1 \leq i \leq m$, $1 \leq j, k \leq n$ such that $v_j \in S'$ and $v_k \notin S'$. This is true because G is connected and in a connected graph every proper subset of vertices will have at least one edge connecting a vertex in the subset to a vertex not belonging to the subset. Let us look at the vertex $e_i \in R_0$. For S' to be a stopping set, e_i must have one more neighbour in S'. But the only possible options are $e'_i \in L_1$ and $v_k \in L_0$. Since we assumed $L_1 \cap S' = \emptyset$, $v_k \in S'$, contradicting our assumption. Hence $L_0 \subseteq S'$.

However, since $L_0 \subseteq S'$, both v_1 and v_2 belong to S'. Since the only neighbours of $z_m \in R_1$ in G_1 (respectively G_2) are $e'_1 \in L_1$ and $v_1 \in L_0$ (respectively $v_2 \in L_0$), it must be true that $e'_1 \in S'$ for satisfying the stopping set condition, thus contradicting $S' \cap L_1 = \emptyset$. Hence $L_1 \subseteq S'$.

The next two lemmas establish the connection between stopping sets in G_1 (or G_2) and vertex covers in G. Recall that a subset S of vertices of G may be identified with the corresponding subset of L_0 in G_1 (or G_2).

Lemma 3. *Let S' be a stopping set in G_1 (respectively G_2) then $S' \cap L_0$ is a vertex cover in G.*

Proof. By Lemma 2, $L_1 \subseteq S'$. Hence by construction $R_0 \subseteq N(S')$. But any $e_i \in R_0$ $1 \leq i \leq m$ has only one neighbour in L_1 and hence must have at least one neighbour in $S' \cap L_0$ in order for S' to satisfy the stopping set condition. This in turn is equivalent to saying that every edge in G has an endpoint in $S' \cap L_0$. Hence $S' \cap L_0$ must be a vertex cover in G.

The following lemma follows directly from the construction of G_1 (respectively G_2).

Lemma 4. *If S is a vertex cover in G then $S' = S \cup L_1 \cup \{v_1\}$ (respectively $S' = S \cup L_1 \cup \{v_2\}$) is a stopping set in G_1 (respectively G_2).*

Proof. Since $L_1 \subseteq S'$, by construction $N(S') = R_0 \cup R_1$. Any $e_i \in R_0$ have $e'_i \in L_1$ as a neighbour and must have at least one neighbour in $L_0 \cap S'$ (because S is a vertex cover). For each $1 \leq i \leq m-1$ z_i has e'_i and e'_{i+1} as neighbours in S'. Finally, e'_1 and v_1 in G_1 (respectively e'_1 and v_2 in G_2) are neighbours of z_m. Hence, every vertex in $N(S')$ has at least two neighbours in S'. Thus S' is a stopping set in G_1 (respectively G_2).

The following lemma relates solutions of $MINVC(3)$ in G and $MINSTOP$ in G_1 and G_2.

Lemma 5. *If the smallest vertex cover in G has size s, $1 \leq s \leq n-1$, then the smallest stopping set in either G_1 or G_2 must be of size at most $s + m$.*

Proof. Let S be a vertex cover of size s in G. Since e_1 must have a neighbour in S, either v_1 or v_2 must belong to S. If $v_1 \in S$ consider G_1 (otherwise consider

G_2). By Lemma 4, $S \cup L_1$ must be a stopping set in G_1. This is the required stopping set of size $s + m$. No stopping set can be smaller because of Lemma 2 and Lemma 3.

We are now ready to prove:

Theorem 1. *If there is a polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ then there exists a polynomial time $(1+4\epsilon)$ approximation algorithm for $MINVC(3)$.*

Proof. Assume that we have a polynomial time $(1+\epsilon)$-approximation algorithm Π for $MINSTOP$. Suppose we are given an instance G of $MINVC(3)$. Let s be the size of a vertex cover of least size in G. Construct G_1 and G_2. Let S_1' and S_2' be the stopping sets in G_1 and G_2 respectively returned by Π. Then by Lemma 5, the smaller of the two stopping sets (say S_1') must have size at most $(1+\epsilon)(s+m)$. By Lemma 2, $L_1 \subseteq S_1$ and hence by Lemma 3 $S_1' \cap L_0 = S_1' \setminus L_1$ must be a vertex cover of size at most $(1+\epsilon)(s+m) - m = s(1+\epsilon) + \epsilon m$ in G. Let us call this vertex cover S. Since every vertex in G has at most 3 edges connected to it, any vertex cover must have size at least $\lceil m/3 \rceil$. Thus we have $m \leq 3s$. Consequently, $|S| \leq (1+4\epsilon)s$. It is clear that this procedure can be completed in polynomial time provided Π runs in polynomial time, yielding the required $(1+4\epsilon)$-approximation algorithm.

The following statement is a direct consequence of Theorem 1.

Corollary 1. *There exists a constant $\epsilon > 0$ such that there is no polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ even for Tanner graphs with left vertex degree bounded by four and right vertex degree bounded by three unless $P = NP$.*

Proof. It is easy to see that G_1 and G_2 constructed above has left degree at most four and right degree at most three if G is an instance of $MINVC(3)$. By [15, p. 369] there exists ϵ_0 such that there is no polynomial time $(1+\epsilon_0)$-approximation algorithm for $MINVC(3)$ unless $P = NP$. Hence, by Theorem 1, existence of a polynomial time $(1+\epsilon_0/4)$-approximation algorithm for $MINSTOP$ would imply $P = NP$.

Our next goal is to show the *self reducibility* of $MINSTOP$ namely, if there is a polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ for some constant $\epsilon > 0$, using this algorithm we can construct another polynomial time $\sqrt{1+\epsilon}$-approximation algorithm for $MINSTOP$. As a consequence, existence of a polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ for any constant $\epsilon > 0$ would imply existence of a polynomial time $(1+\epsilon)$-approximation for $MINVC(3)$ for *every* constant $\epsilon > 0$ which is impossible unless $P = NP$.

3.2 Self-reducibility of MINSTOP

Suppose we are given an instance of $MINSTOP$ — an undirected bipartite graph $G = (L, R, E)$. Let $L = \{x_1, .., x_n\}$, $R = \{c_1, ..., c_m\}$. We construct an undirected bipartite graph $G' = (L', R', E')$ such that $L' = L_1 \cup ... \cup L_n$, $R' = R_0 \cup ... \cup R_n$, $E' = E_0 \cup ... \cup E_n$, where for $1 \leq k \leq n$ $L_k = \{x_i^k, 1 \leq i \leq n\}$, $R_k = \{c_j^k, 1 \leq j \leq m\}$, $E_k = \{(x_i^k, c_j^k) : (x_i, c_j) \in E\}$, $R_0 = \{z_{ij} : 1 \leq i < j \leq n\}$ and $E_0 = \{(x_i^j, z_{ij}), (z_{ij}, x_j^i) : 1 \leq i < j \leq n\}$. Figure 3 illustrates the construction for the Tanner graph in Figure 1.

The following description would help the reader to develop a better intuition about the structure of G'. G' consists of n copies of the Tanner graph G, where the kth copy is the subgraph of G' induced by vertices L_k and R_k and contains all the edges in the set E_k. We shall denote this subgraph as G'_k. The vertex $x_i \in L$ appears as $x_i^k \in L_k$, $c_j \in R$ appears as $c_j^k \in R_k$ and the connections established by E_k are identical to those by E in G. The connections between different copies are defined by the edges in E_0 and the vertices in R_0 are used as intermediate vertices in the connection. The vertex x_i^k in L_k is connected to x_k^i in L_i through the vertex z_{ik}. This connection guarantees that any stopping set S' in G' that contains x_i^k must also contain x_k^i for otherwise the vertex z_{ik} will have only one neighbour in S' violating the stopping set condition.

To simplify the notation the following convention will be used. We shall identify the vertex $x_i^k \in L_k$ with $x_i \in L$, $1 \leq i \leq n$, and $c_j^k \in R_k$ with $c_j \in R$ $1 \leq j \leq m$. Then we have a natural isomorphism between G and G'_k for each $1 \leq k \leq n$. Let S' be a stopping set in G'. We use the notation S'_k to denote the set $S' \cap L_k$. We then have $S' = S'_1 \cup ... \cup S'_n$ and the union is disjoint. We refer to S'_k as a subset of L in G using the above identification without mention of that fact when there is no confusion.

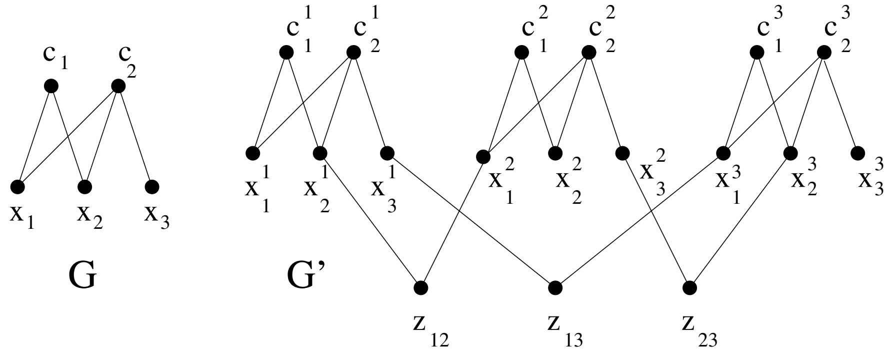


Fig. 3. Construction of G - example illustrating self-reducibility of MINSTOP

The following lemma summarizes a key property of stopping sets in G'.

Lemma 6. *Let S' be a stopping set in G'. Let $S'_k \neq \emptyset$ for some $1 \leq k \leq n$. Then S'_k is a stopping set in G (under the identification above).*

Proof. Since vertices in R_k have neighbours only in the set L_k in G', any vertex in R_k with a neighbour in S'_k must have at least one more neighbour in S'_k in order for S' to satisfy the stopping set condition. Thus, S'_k must be a stopping set in the subgraph of G' induced by L_k and R_k namely, G'_k. Since G'_k is isomorphic to G under our identification, S'_k must be a stopping set in G.

The next lemma proves that from any given stopping set S' in G' we can extract a stopping set S in G such that $|S'| \geq |S|^2$.

Lemma 7. *Let S' be a stopping set in G'. Let $H = \{S'_k : S'_k \neq \emptyset, 1 \leq k \leq n\}$. Let S'_i be an element of H of minimum cardinality. Then $|S'_i| \leq \sqrt{|S'|}$.*

Proof. Let $t = |S'_i|$. Without loss of generality we may assume that $S'_i = \{x^i_1, ..., x^i_t\}$ (otherwise re-label the vertices). Since z_{ij} (if $i > j$ read z_{ji}) which is a neighbour of x^i_j must have at least two neighbours in S' to satisfy the stopping set condition, $x^j_i \in S'$ for $1 \leq j \leq t$. Hence $S'_j \in H$ for $1 \leq j \leq t$. By the assumption that S'_i is a member in H of least cardinality, $|S'_j| \geq |S'_i|$ for $1 \leq j \leq t$. Since every $S'_j \in H$ is contained in S' and are disjoint, $|S'| \geq t|S'_i| = t^2$.

Now we show that for every stopping set S in G, we can find a stopping set in G' containing $|S|^2$ vertices.

Lemma 8. *Let S be a stopping set in G then $S' = \{x^j_i : x_i \in S$ and $x_j \in S\}$ is a stopping set of size $|S|^2$ in G'.*

Proof. First, note that it follows from the definition of the set S' that if $x_i \in S$ then $S'_i = S$ (under our identification) and if $x_i \notin S$ then $S'_i = \emptyset$ for all $1 \leq i \leq n$. Let y be a neighbour of a vertex in S'. Then by the definition of S', either $y \in R_i$ for some $i \in \{1, ...n\}$ such that $x_i \in S$ or $y \in R_0$. In the first case, since S'_i is a stopping set in G (because $S'_i = S$ under our identification and S is a stopping set in G), y must have at least two neighbours in S'_i and hence in S'. In the latter case, $y = z_{ij}$ for some $1 \leq i < j \leq n$ and both x^j_i and x^i_j are neighbours of y in S'. Thus in all cases the stopping set condition is satisfied.

As a result of all the above we have:

Lemma 9. *Let S be a stopping set of minimum size in G then the stopping set of minimum size in G' has size $|S|^2$.*

Proof. Let S' be a stopping set in G' consider the set $H = \{S'_k : S'_k \neq \emptyset, 1 \leq k \leq n\}$ defined in Lemma 7. By Lemma 6, each element $S'_k \in H$ must be a stopping set in G and therefore $|S'_k| \geq |S|$ for each $S'_k \in H$ as S is a stopping set of minimum size in G. Hence By Lemma 7, $|S'| \geq |S|^2$. Existence of a stopping set of size $|S^2|$ follows from Lemma 8.

Theorem 2. *If there is a polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ for any constant $\epsilon > 0$ then there exists a polynomial time $\sqrt{1+\epsilon}$-approximation algorithm for $MINSTOP$.*

Proof. Suppose there is a polynomial time $(1+\epsilon)$-approximation algorithm Π for $MINSTOP$. Given an instance G of $MINSTOP$, construct G' and run Π

on input G'. Let s be the size of a stopping set of smallest size in G. By Lemma 9, the size of any stopping set of smallest size in G' must be s^2. Thus Π must return a stopping set S' of size at most $(1+\epsilon)s^2$ in G'. Hence by Lemma 7 we can find an S'_i of size at most $(\sqrt{1+\epsilon})s$ in G' which by Lemma 6 must be a stopping set in G. Since G' has size of the order of square the size of G, the procedure runs in polynomial time provided Π runs in polynomial time.

We collect the non-approximability results that follow as consequences of Theorem 1 and Theorem 2 in the following sub-section. We shall deviate from our earlier notation and use the variable n here to denote the "size" of a graph. (Size of a graph is normally measured in terms of the number of bits needed to represent the graph using an adjacency matrix/list.)

3.3 Non-approximability of MINSTOP

Theorem 3. *There exists no polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ for any constant $\epsilon > 0$ unless $P = NP$.*

Proof. Suppose there is a polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ for some constant $\epsilon > 0$. Fix any constant $\epsilon_0 < \epsilon$. Let $r = \left\lceil \log_2(\log_{(1+\epsilon_0)}(1+\epsilon)) \right\rceil$. Given an instance G of $MINSTOP$ of size n, by an r-fold repeated application of self-reduction on G we get a graph G' of size $O(n^{2^r})$ from which we can extract a stopping set of size at most $(1+\epsilon)^{2^{-r}}$ times the size of the smallest stopping set in G. Note that r is a constant independent of n and hence the procedure will run in time polynomial in n. By our choice of r, $(1+\epsilon)^{2^{-r}} < (1+\epsilon_0)$ and thus the procedure yields a polynomial time $(1+\epsilon_0)$-approximation algorithm for $MINSTOP$. Note that the procedure works with any constant $\epsilon_0 > 0$. By Corollary 1, there is some constant $\epsilon_0 > 0$ for which this is impossible unless $P = NP$. Hence existence of a polynomial time $(1+\epsilon)$-approximation algorithm for $MINSTOP$ for any constant $\epsilon > 0$ implies $P = NP$.

The above proof may be extended to yield a stronger non-approximability assertion if we assume a hypothesis weaker than $P \neq NP$.

Theorem 4. *There exists no polynomial time $2^{(\log n)^{1-\epsilon}}$-approximation algorithm for $MINSTOP$ for any constant $\epsilon > 0$ unless $NP \subseteq DTIME(n^{poly(\log n)})$.*

Proof. Suppose there is a polynomial time $2^{(\log n)^{1-\epsilon}}$-approximation algorithm for $MINSTOP$ for some constant $\epsilon > 0$. Given an instance G of $MINSTOP$ of size n and any constant $\epsilon_0 > 0$, let $r = \lceil (1/\epsilon)(\log_2 \log_2 n - \log_2 \log_2(1+\epsilon_0)) \rceil$. As with the proof of Theorem 3, an r-fold application of the self reducibility of $MINSTOP$ on G results in a graph G' of $O(n^{2^r}) = O(n^{poly(\log n)})$ size from which we can extract a stopping set of size at most $(1+\epsilon_0)$ times the size of the smallest stopping set in G. By Corollary 1, this would yield an $O(n^{poly(\log n)})$ algorithm for $MINVC(3)$ which in turn would imply that every NP-optimization problem would be in $DTIME(n^{poly(\log n)})$.

4 Discussion and Conclusion

There is an extensive literature on complexity results on problems in coding theory — particularly on the problem of finding the *minimum distance* of a linear code [17,18,19,20,21] which is closely related to $MINSTOP$. It can be shown that the self-reducibility proof for stopping distance presented in this paper is adaptable to yield a proof for self-reducibility of the problem of finding the minimum distance of a binary linear code, although the resultant non-approximability result is slightly weaker than the best known in literature [20]. The question of whether there exist a constant $\epsilon > 0$ such that there is a polynomial time n^{ϵ} approximation algorithm for $MINSTOP$ is open for further investigation.

Acknowledgment

The authors would like to thank Prof. Priti Shankar for helpful discussions and suggestions.

References

1. C. Di, D. Proietti, I. E. Telatar, T. J. Richardson and R. Urbanke, “Finite length analysis of low-density parity-check codes on the binary erasure channel,” *IEEE Trans. Inform. Theory.*, vol. 48, no. 6, pp. 1570-1579, June 2002.
2. M. Tanner, “A recursive approach to low-complexity codes,” *IEEE Trans. Inform. Theory*, vol. 27, no. 5 pp. 533-547, Sept 1981.
3. C. Di, A. Montanari and R. Urbanke, “Weight distribution of LDPC code ensembles: Combinatorics meets statistical physics,” in *Proc. IEEE Int. Symp. Inform. Theory*, Chicago, IL., July 2004, p. 102.
4. A. Orlitsky, K. Viswanathan, and J. Shang, “Stopping set distribution of LDPC code ensembles,” *IEEE Trans. Inform. Theory*, vol. 51, no. 3, March 2005, pp. 929-953.
5. T. Tian, C. Jones, J. D. Villasenor, and R. D. Wesel, “Construction of irregular LDPC codes with low error floors,” in *Proc. ICC 2003*, Seattle, Washington, May 2003, pp. 3125-3129.
6. A. Ramamoorthy, and R. Wesel, “Construction of short block length irregular LDPC codes,” in *Proc. ICC 2004*, Paris, June 2004, pp. 410-414.
7. M. Schwartz and A. Vardy, “On the stopping distance and the stopping redundancy of codes,” *IEEE Trans. Inform. Theory* vol. 52, no. 3, pp. 922-932, March 2006.
8. M. R. Garey, and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W. H. Freeman, 1979.
9. H. Pishro-Nik and F. Fekri, “On decoding of low-density parity-check codes over the binary erasure channel,” *IEEE Trans. Inform. Theory*, vol. 50, no. 3, pp. 439-454, March 2004.
10. J. Han and P. Siegel, “Improved upper bounds on stopping redundancy,” preprint available at: *http://www.arXiv.org*, cs.IT/0511056.
11. R. G. Gallager, ”Low density parity-check codes”, MIT Press, 1963.

12. K. Murali Krishnan and Priti Shankar, “On the complexity of finding stopping distance in Tanner graphs,” preprint available at: *http://www.arXiv.org*, cs.IT/0512101.
13. M. Artin, ”Algebra,” Prentice Hall, 1991.
14. V. V. Vazirani, ”Approximation Algorithms,” Springer, 2004.
15. G. Ausiello et. al., ”Complexity and approximation: combinatorial optimization problems and their approximability properties,” Springer, 2003.
16. D. Hochbaum (Ed.), ”Approximation algorithms for NP-hard problems,” Course Technology, 1996.
17. I. Dumer, D. Micciancio and M. Sudan, “Hardness of approximating the minimum distance of a linear code,” *IEEE Trans. Inform. Theory*, vol. 49, no. 1, pp. 475-484, Jan. 2003.
18. A. Vardy, “The intractability of computing the minimum distance of a code,” *IEEE Trans. Inform. Theory*, vol. 43, no. 6, pp. 1757-1766, Nov. 1997.
19. E. Berlekamp, R. McEliece and H. van Tilborg, “On the inherent intractability of certain coding problems,” *IEEE Trans. Inform. Theory*, vol. 24, no. 3, pp. 384-386, May. 1978.
20. I. Dinur, G. Kindler and S. Safra, “Approximating CVP to within almost polynomial factors in NP-hard,” *Proc. FOCS 1998*, Palo Alto, California, Nov. 1998, pp. 99-111.
21. S. Arora, L. Babai, J. Stern and E. Z. Sweedyk, “The hardness of approximate optima in lattices, codes and systems of linear equations,” *J. Comput. Sys. Sci.*, vol. 54, no. 2, pp. 317-331, April. 1997.

Multi-stack Boundary Labeling Problems*

Michael A. Bekos[1], Michael Kaufmann[2],
Katerina Potika[1], and Antonios Symvonis[1]

[1] National Technical University of Athens,
School of Applied Mathematical & Physical Sciences
{mikebekos, symvonis}@math.ntua.gr, epotik@cs.ntua.gr
[2] University of Tübingen, Institute for Informatics
mk@informatik.uni-tuebingen.de

Abstract. The *boundary labeling* problem was recently introduced in [5] as a response to the problem of labeling dense point sets with large labels. In boundary labeling, we are given a rectangle R which encloses a set of n sites. Each site is associated with an axis-parallel rectangular label. The main task is to place the labels in distinct positions on the boundary of R, so that they do not overlap, and to connect each site with its corresponding label by non-intersecting polygonal lines, so called *leaders*. Such a label placement is referred to as *legal label placement*.

In this paper, we study boundary labeling problems along a new line of research. We seek to obtain labelings with labels arranged on more than one stacks placed at the same side of R. We refer to problems of this type as *multi-stack boundary labeling problems*.

We present algorithms for *maximizing the uniform label size* for boundary labeling with two and three stacks of labels. The key component of our algorithms is a technique that combines the merging of lists and the bounding of the search space of the solution. We also present NP-hardness results for multi-stack boundary labeling problems with labels of variable height.

1 Introduction

A common task in the process of information visualization is the placement of extra information, usually in the form of text labels, next to the features of a drawing (diagram, map, technical or graph drawing). When the labels are small and the features are sparsely distributed in the drawing, it may be feasible to place most labels next to the features so that the labels do not overlap with each other and they do not obscure other drawing features. Obtaining optimal label placements with respect to some optimization criterion is, in general, NP-hard [8]. An extensive bibliography about map labeling can be found at [12].

In the case of very large labels (or, equivalently, dense feature sets), it is usually impossible to find a label placement, i.e. to place each label next to the feature.

* The work is co - funded by the European Social Fund (75%) and National Resources (25%) - Operational Program for Educational and Vocational Training II (EPEAEK II) and particularly the Program PYTHAGORAS.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 81–92, 2006.

In response to this problem, Bekos, Kaufmann, Symvonis and Wolff [5] (an extended journal version appears in [4]) proposed the *boundary labeling model.* In this model the labels are placed on the boundary of a rectangle enclosing all features and each label is connected to its associated feature with polygonal lines, called *leaders.* If the labels are non overlapping and the leaders non intersecting we have a *legal labeling* or a *legal label placement.* The boundary labeling model is a realistic model for medical atlases and technical drawings, where certain features of a drawing are explained by blocks of text placed outside the drawing so that no part of the drawing is obscured. SmartDraw [10] provides boundary labelings in a primitive form based on labeling templates. It does not support any form of automated boundary labeling optimization. Bler [6] supports the boundary labeling process and facilitates the annotation of drawings with text labels.

Sites model features of the drawing. If they model a *point-feature* (e.g., a city on a map) they are naturally represented as points (see points in rectangle R of Fig. 1, 2 and 5). So, in its simplest form, a boundary labeling problem specifies as part of its input a set P of n points $p_i = (x_i, y_i)$ on the plane in general position, i.e. no three points lie on a line and no two points have the same x- or y-coordinate. Another interesting variation is the one with two candidate points on the plane for each site (see Fig. 3). In practice, several times we want to associate a label with an *area-feature* (e.g., a region on a map). To keep things simple, we specify these regions by a closed polygonal line or by a line segment internal to the feature area, and assume that the site "slides" along the boundary of the polygon or on the line segment (see Fig. 4).

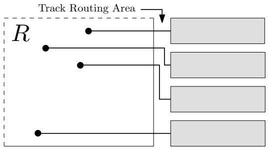


Fig. 1. Type-*opo* leaders

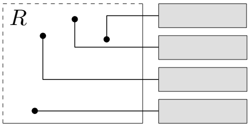


Fig. 2. Type-*po* leaders

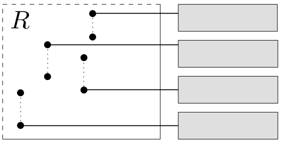


Fig. 3. Sites with 2 candidate points

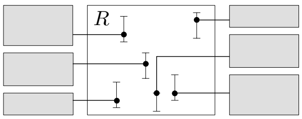


Fig. 4. Sites are vertical line segments

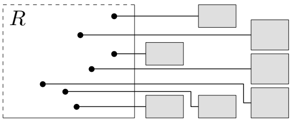


Fig. 5. Three stacks of labels

In general, each site p_i has a corresponding axis-parallel rectangular, open label l_i of width w_i and height h_i. The labels are to be placed around an axis-parallel rectangle $R = [l_R, r_R]\text{x}[b_R, t_R]$ of height $H = t_R - b_R$ and width $W = r_R - l_R$ which contains all sites $p_i \in P$. While in the general case the labels are of *variable dimensions*, we also consider the restricted cases where the labels are of *uniform size* (height and/or width), or of *maximum uniform size.*

Each site is connected with its corresponding label in a simple and elegant way by using polygonal lines, called *leaders*. In our approach we have leaders that consist of a single straight line segment or a sequence of rectilinear segments. When a leader is rectilinear, it consists of a sequence of axis-parallel segments either parallel (p) or orthogonal (o) to the side of R containing the label it leads to. The *type* of a leader is defined by an alternating string over the alphabet $\{p, o\}$. We focus on leaders of types-*opo* and *po*, see Fig. 1 and 2, respectively. Furthermore, we assume that each type-*opo* leader has its parallel p-segment (or equivalently its both bends) outside R, routed in the so-called *track routing area*. We consider type-*o* leaders to be of type-*opo* and of type-*po* as well.

A further refinement of the labeling model has to do with the sides of the enclosing rectangle containing the labels. Labels can be placed on one or more sides of R (in Fig. 1, 2, 3 and 5 all labels are placed on the east side of R). In order to allow for *greater numbers* of *larger labels*, we might have the labels arranged in more than one stack at each side of the enclosing rectangle. This paper is devoted to the case of *multi-stack labelings*. Figure 5 shows a labeling where the labels occupy three stacks to the east side of R. Notice that in the case of multiple stacks of labels (say m stacks), a leader of type-*opo* can have its p segment either in between R and the first stack (called *first track routing area*) or between the i-th and the $(i+1)$-th stack, where $i < m$ (called $(i+1)$*-th track routing area*).

Each leader that connects a site to a label, touches the label on a point on its side that faces R, this point is called *port*. We can assume either *fixed ports*, i.e. the leader is only allowed to use a fixed set of ports on the label side (a typical case is where the leader uses the middle point of the label side) or *sliding ports* where the leader can touch any point of the label's side. The labelings in Fig. 1, 2 and 5 use fixed ports, while in Fig. 3 and 4 they use sliding ports.

Keeping in mind that we want to obtain simple and easy to visualize labelings, the following criteria can be adopted from the areas of VLSI and graph drawing: *minimizing the total number of bends* of the leaders, *minimizing the total leader length* and *minimizing the maximum leader length*. An additional criterion that we consider is the *maximization of the uniform label size*. This is a quite common optimization criterion in the map labeling literature. In this paper, we seek to obtain labelings with labels of maximum uniform size arranged on more than one stacks of labels at the same side of R.

This paper is structured as follows: Section 2, reviews preliminary results required for the development of our algorithms. In Section 3, we present algorithms for obtaining multi-stack labelings of maximum uniform label size for the cases of two and three stacks of labels arranged at the same side of R. In Section 4, we present several NP-hardness results for non-uniform labels placed in two stacks. We conclude in Section 5 with open problems and future work.

Previous Work

Most of the known results on boundary labeling with point sites were presented in [4]. A legal labeling, on one side with type-*opo* (type-*po*) leaders can be achieved in $O(n \log n)$ time (in $O(n^2)$ time, respectively), whereas on all four sides with type-*opo* leaders in $O(n \log n)$ time. The problem of minimizing the total number of leader bends on one side with type-*opo* leaders can be solved in $O(n^2)$ time. The minimization of the total leader length when uniform sized labels can be placed on two opposite sides of R with either type-*opo* and type-*po* leaders needs $O(n^2)$ time. For the similar problem, where non-uniform labels can be placed on two opposite sides of R and the leaders are of type-*opo*, $O(nH^2)$ time is needed. An algorithm for minimizing the total leader length on four sides with type-*opo* leaders in $O(n^2 \log^3 n)$ ($O(n^3)$) time for fixed ports (sliding ports) is presented for points in [1] and for polygons in [3].

2 Preliminaries

Throughout the paper we use lists that contain pairs of integers describing different label placements. Given a pair (a, b) of integers, a and b are referred to as the *first* and the *second coordinate* of the pair, respectively. Inspired by an idea of Stockmeyer [11] which was subsequently used by Eades et. al. [7], we manage to keep the length of each list bounded by pruning pairs that cannot occur in an optimal solution.

Definition 1. *A list L of pairs of integers is* ***sorted*** *if the pairs it contains are lexicographically sorted in decreasing order with respect to their first coordinate and in increasing order with respect to their second coordinate.*

Definition 2. *Let (a, b) and (c, d) be pairs of integers.*

$$(a, b)\ \textbf{\textit{dominates}}\ (c, d) \iff a \geq c \text{ and } b \geq d.$$

Suppose that we have to solve a problem where the search space of the solution consists of pairs of integers, and let f be a monotone function computing a minimization objective on pairs from the solution search space. If (a, b) and (c, d) represent possible solutions and (a, b) dominates (c, d), then the pair (a, b) can never be involved in an optimal solution and may be safely removed from the solution set. Given a list L of pairs of integers, a pair $(a, b) \in L$ that does not dominate any other pair in L is called an *atom* (with respect to L).

In our algorithms we maintain lists (of pairs) that contain only atoms. A frequently performed operation is the merging of two lists of atoms, resulting in a new list of atoms. The merging algorithm resembles the merging step of merge sort algorithm. It supports the following lemmas:

Lemma 1. *k sorted lists $L_1, L_2, \ldots, L_k$, $k \geq 2$, of atoms can be merged in $O((k-1)\sum_{i=1}^{k} |L_i|)$ time into a new sorted list L of at most $\sum_{i=1}^{k} |L_i|$ atoms.*

Lemma 2. *Let A and B be two finite sets of integers and let $L = \{(a, b) |\ a \in A \text{ and } b \in B\}$ be a list of atoms. Then, $|L| \leq \min(|A|, |B|)$.*

Finally, we present some notation and terminology that we use in the description of our algorithms. We say that a pair (a, b) obeys the *boundary conditions*, if

$a \leq H$ and $b \leq H$, where H is the height of the enclosing rectangle. We also define operator $\oplus_H : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$, where:

$$a \oplus_H b = \begin{cases} a+b, \text{ if } a+b \leq H \\ \infty, \quad \text{otherwise} \end{cases} .$$

3 Label Size Maximization

3.1 Two Stacks of Labels on the Same Side

We consider boundary labeling with type-*opo* leaders, where the labels are placed at two stacks on the same side (say, the east side) of the rectangle R. We assume that all labels have the same size (width and height) and we seek to maximize the uniform height h of all labels, so that a legal labeling exists. To determine the maximum value of h, we apply a binary search on all possible discrete values for height h. We assume the more general case of sliding ports. Additionally, the type-*opo* leaders connecting sites to labels that are at the second stack are allowed to bend either in the first or in the second track routing area.

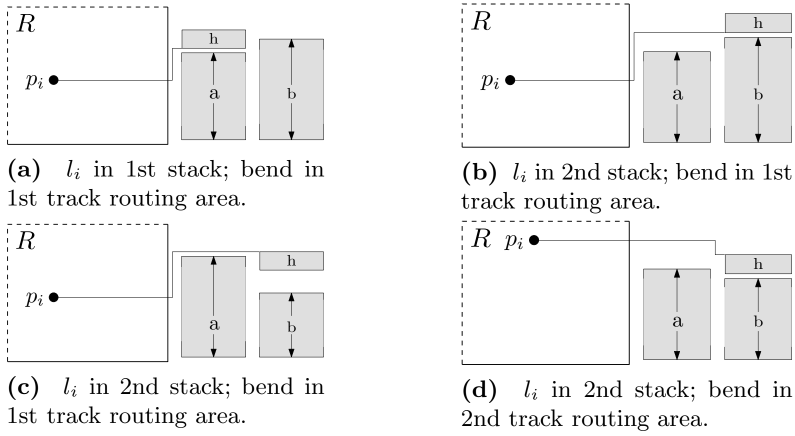
Observe that, in any legal one-side labeling with type-*opo* leaders, the vertical order of the sites is identical to the vertical order of their corresponding labels on both stacks. This, together with the assumption that no two sites share the same y-coordinate, guarantees that leaders do not intersect. So, we assume that the sites are sorted according to increasing y-coordinate.

For a fixed h, we propose a dynamic programming algorithm that outputs a boolean value, which indicates whether there exists a legal label placement, when all sites have labels of height h. Imagine that a label placement L is given, then we say that a pair (a, b) *describes* L, if a (b) is the highest occupied y-coordinate of the first (respectively second) stack. Our algorithm maintains a table T of size $(n+1) \times (n+1)$, where each entry $T[i, k]$, $i \geq k$, of table T contains a list of atoms (a, b) decribing the label placement of the first i sites when k out of them have leaders bending in the second track routing area. List $T[i, k]$ is empty, when it is impossible to place the first i labels, with k leaders bending in the second track routing area.

Assuming that we have placed the labels for the first $i-1$ sites, we try to place the label l_i of the i-th site. Label l_i can be placed at the first or second stack. Additionally, if l_i is to be placed at the second stack, then we have to check whether this can be done with a leader bending in the first or second track routing area. Obviously, such placements can be obtained from label placements of the first $i-1$ sites with either k or $k-1$ leaders bending in the second track routing area.

Label l_i is placed at the first stack: Let $T_1[i, k]$ be a list of pairs (a, b) describing the label placement of the first i sites when the i-th site has its label at the first stack and k leaders have their bends in the second track routing area. $T_1[i, k]$ can be computed based on entry $T[i-1, k]$ (see Fig. 6a), as follows:

$$T_1[i, k] = \{(a \oplus_H h, b) : \ \forall (a, b) \in T[i-1, k]\}$$



(a) l_i in 1st stack; bend in 1st track routing area.

(b) l_i in 2nd stack; bend in 1st track routing area.

(c) l_i in 2nd stack; bend in 1st track routing area.

(d) l_i in 2nd stack; bend in 2nd track routing area.

Fig. 6. Different placements obtained for the label of site i. In Figures 6a, 6b and 6c: $(a, b) \in T[i-1, k]$, whereas in Fig. 6d: $(a, b) \in T[i-1, k-1]$.

Label l_i is placed at the second stack - bend at the first track routing area: Let $T_{21}[i, k]$ be a list of pairs (a, b) describing the label placement of the first i sites when the i-th site has its label at the second stack using a leader bending at the first track routing area and k leaders have their bends in the second track routing area. Again, $T_{21}[i, k]$ can be computed based on entry $T[i-1, k]$. If for some pair $(a, b) \in T[i-1, k]$ it holds that $a \leq b$ (i.e. the first stack is lower or equal than the second stack), then a pair $(b, b \oplus_H h)$ is added in $T_{21}[i, k]$ (see Fig. 6b). Else pair $(a, \max\{b \oplus_H h, a\})$ is added in $T_{21}[i, k]$ (see Fig. 6c). Therefore, $T_{21}[i, k]$ is computed as follows:

$$T_{21}[i, k] = A_{21}[i, k] \cup B_{21}[i, k],$$

where:

$$A_{21}[i, k] = \{(b, b \oplus_H h) : \forall (a, b) \in T[i-1, k] \text{ s.t. } a \leq b\}$$
$$B_{21}[i, k] = \{(a, \max\{b \oplus_H h, a\}) : \forall (a, b) \in T[i-1, k] \text{ s.t. } a > b\}$$

Label l_i is placed at the second stack - bend at the second track routing area: Let $T_{22}[i, k]$ be a list of pairs (a, b) describing the label placement of the first i, when the i-th site has its label placed at the second stack using a leader bending at the second track routing area and k leaders have their bends in the second track routing area. $T_{22}[i, k]$ is computed based on entry $T[i-1, k-1]$ (see Fig. 6d), as follows:

$$T_{22}[i, k] = \{(y_i, b \oplus_H h) : \forall (a, b) \in T[i-1, k-1] \text{ s.t. } a < y_i\}$$

All pairs (∞, a), (a, ∞) can be removed from lists $T_1[i, k]$, $T_{21}[i, k]$ and $T_{22}[i, k]$, in linear time, since they do not capture possible placements. The implied lists are merged into list $T[i, k]$ of atoms, based on Lemma 1. We can easily show that $|T[i, k]| \leq 2|T[i-1, k]| + 3$. This implies that $|T[n, k]| = O(2^n)$, $n \geq k$. Also, by Lemma 2, we have that $|T[n, k]| \leq H$. However, by employing the following

Lemma 3, we can improve on both of these bounds. Its correctness can easily be shown inductively, by proving that the distinct values that both coordinates of the pairs in $T[i,k]$ can receive are drawn from the sets $\{0, h, 2h, \ldots, ih\}$, $\{y_1, y_2, \ldots, y_i\}$, and $\bigcup_{j=1}^{i}\{y_j + h, y_j + 2h, \ldots, y_j + (i-1)h\}$.

Lemma 3. *List $T[n,k]$, $n \geq k$ contains $O(n^2)$ pairs.*

To prove the correctness of our algorithm, consider a pair $(a,b) \in T[i,k]$ that dominates pair $(c,d) \in T[i,k]$. Assume, for the sake of contradiction, that pair (a,b) yields a solution and pair (c,d) does not. That means that, for at least one pair out of $\{(y_i, b+h), (b, b+h), (a, \max\{b+h, a\}), (a+h, b)\}$ the boundary condition holds while the boundary condition does not hold for any of the pairs $\{(y_i, d+h), (d, d+h), (c, \max\{d+h, c\}), (c+h, d)\}$. This is impossible since $a \geq c$ and $b \geq d$. Therefore (a,b) can never be involved in an optimal solution and can be discarded. This implies that each list $T[i,k]$ should only contain atoms.

Each of the $(n+1) \times (n+1)$ entries of T is computed in $O(n^2)$ time. Thus, our algorithm terminates after $O(n^4)$ time. For a fixed label height h, the algorithm outputs a boolean value, which indicates whether there exists a legal label placement. This is done by identifying whether there exists a non-empty list $T[n,j]$, with $0 \leq j \leq n$. By using an extra table of the same size as T, our algorithm can easily be modified, such that it also computes the label and leader positions.

Theorem 1. *Given a rectangle R of integer height H and a set $P \subset R$ of n points in general positions, there exists an $O(n^4 \log H)$ time algorithm that produces a legal multi-stack labeling with two stacks of labels on the same side of R and with type-opo leaders such that the uniform integer height of the labels is maximum.*

Proof. In order to solve the *label size maximization problem*, we can simply apply a binary search on all possible discrete values for height h. To complete the proof, observe that $\frac{H}{n} \leq h \leq \frac{2H}{n}$. □

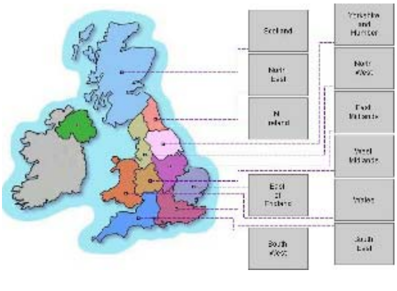


Fig. 7. A regional map of UK

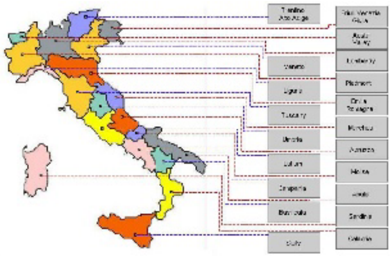


Fig. 8. A regional map of Italy

Sample Labelings
Figures 7 and 8 are produced from the algorithm of Section 3.1 and depict two regional maps of UK and Italy, respectively. The labels occupy two stacks on the east side of the enclosing rectangle. In both labelings the label size is maximum.

3.2 Three Stacks of Labels on the Same Side

In this section, we extend the algorithm of Section 3.1 to support an additional stack of labels. We consider the case, where leaders connected to labels at the i-th stack are restricted to bend in the i-th track routing area. The objective, again, is to maximize the uniform height h of all labels.

Theorem 2. *Given a rectangle R of integer height H and a set $P \subset R$ of n points in general positions, there exists an $O(n^4 \log H)$ time algorithm that produces a legal multi-stack labeling with three stacks of labels on the same side of R and with type-opo leaders such that the uniform integer height of the labels is maximum and the leaders connected to labels at the i-th stack are restricted to bend in the i-th track routing area.*

Proof. We use dynamic programming algorithm employing a table T of size $(n+1) \times (n+1) \times (n+1)$. For each $i \geq k+m$, entry $T[i,k,m]$ contains a list of pairs (a,b), where a (b) is the y-coordinate of the first (second) stack, that is needed to place the first i labels, when m labels are placed in the third stack, k labels are placed in the second stack and $i-k-m$ labels are placed in the first stack. Note that the height at the third stack is mh, since all leaders connected to labels of the third stack are restricted to bend in the third track routing area. List $T[i,k,m]$ is empty, when it is impossible to route the first i labels using k labels in the second stack and m in the third stack. This implies that table entries $T[i,k,m]$, where $i < k+m$, contain empty lists. Following similar arguments as in Section 3.1, entry $T[i,k,m]$ can be computed based on the following recurrence relation:

$$T[i,k,m] = \textsc{Merge}\{T_1[i,k,m], T_2[i,k,m], T_3[i,k,m]\} \tag{1}$$

where:

$$T_1[i,k,m] = \{(a \oplus_H h, b) : \ \forall (a,b) \in T[i-1,k,m]\}$$
$$T_2[i,k,m] = \{(y_i, b \oplus_H h) : \ \forall (a,b) \in T[i-1,k-1,m] \text{ s.t. } a < y_i\}$$
$$T_3[i,k,m] = \{(y_i, y_i) : \ \forall (a,b) \in T[i-1,k,m-1], \text{ s.t. } mh \leq H \text{ and } (a,b) < (y_i, y_i)\}$$

List $T_1[i,k,m]$ of Eq. 1 captures placements of the i-th label at the first stack. Similarly, list $T_2[i,k,m]$ of Eq. 1 captures placements of the i-th label at the second stack. Since we assumed that leaders connected to labels at the second stack are restricted to bend in the second track routing area, this is possible only for pairs $(a,b) \in T[i-1,k-1,m]$ with $a \leq y_i$. Finally, list $T_3[i,k,m]$ of Eq. 1 captures placements of the i-th label at the third stack. This is possible only for pairs $(a,b) \in T[i-1,k,m-1]$ with $(a,b) \leq (y_i, y_i)$. To compute entry $T[i,k,m]$, we first remove all pairs (∞, a), (a, ∞) from lists $T_1[i,k,m]$, $T_2[i,k,m]$

and $T_3[i,k,m]$ and then we merge the implied lists to $T[i,k,m]$ of atoms, based on Lemma 1.

Lemma 4. *For $n \geq k+m$, $|T[n,k,m]| \leq n+1$.*

Proof. Lists $T_2[i,k,m]$ and $T_3[i,k,m]$ contain pairs of numbers with the same first coordinate. This implies that they contribute at most one atom, while list $T_1[i,k,m]$ contains at most i elements, since $|T[i-1,k,m]| \leq i$. Thus, $T[i,k,m] \leq i+1$. □

Each of the $(n+1) \times (n+1) \times (n+1)$ entries of T is computed in $O(n)$ time. Thus, our algorithm terminates after $O(n^4)$ time. For a fixed label height h, the algorithm outputs a boolean value, which indicates whether there exists a legal label placement. This is done by identifying whether there exists a non-empty list $T[n,i,j]$, with $0 \leq i+j \leq n$. By using an extra table of the same size as T, our algorithm can easily be modified, such that it also computes the label and leader positions. In order to solve the *label size maximization problem*, we can simply apply a binary search on all possible discrete values for height h. To complete the proof, observe that $\frac{H}{n} \leq h \leq \frac{3H}{n}$. □

4 Computational Complexity of the Multi-stack Labeling Problem

In this section, we investigate the computational complexity of several multi-stack boundary labeling problems with either type-*opo* or *po* leaders and labels of arbitrary size, which can be placed at two stacks on the same side of the enclosing rectangle. Without loss of generality, we assume that the labels are located on the east side of the enclosing rectangle. We consider several different type of sites. In the most applicable case, site s_i is associated with a point $p_i = (x_i, y_i)$ on the plane. However, we also consider the cases, where site s_i is associated with either two candidate points $p_i^1 = (x_i^1, y_i^1)$ and $p_i^2 = (x_i^2, y_i^2)$ on the plane (see Fig. 3) or with a vertical line segment, so that the site "slides" along the boundary of the proposed line segment (see Fig. 4). The assumed models are quite general, since we allow sliding labels with sliding ports.

4.1 Line Sites with Type-*opo* Leaders at Two Stacks on One Side

We focus on type-*opo* leaders, where each site s_i can slide along a line segment parallel to the y-axis and is associated with a label l_i of height h_i. We seek to find a legal labeling.

Theorem 3. *Given a rectangle R of height H, a set $P \subset R$ of n line segments (sites) that are parallel to the y-axis and a label of height h_i for each site $s_i \in P$, it is NP-hard to place all labels at two stacks on one side of R with non-intersecting type-opo leaders.*

Proof. We reduce the PARTITION problem [9] to our problem. The PARTITION problem is defined as follows: Given positive integers $a_1, a_2, \ldots, a_m$, is there a subset I of $J = \{1, 2, \ldots, m\}$ such that $\sum_{i \in I} a_i = \sum_{i \in J-I} a_i$?

Our site set $P = \{s_1, s_2, \dots, s_m\}$ consists of m (parallel to y-axis) line segments of identical length $H = \frac{1}{2}\sum_{i\in J} a_i$ (H: height of R). Each site s_i is also associated with a label l_i of height a_i. Both stacks contribute $2H$ height, which is equal to the sum of all label heights.

Suppose that there exists a subset I of $J = \{1, 2, \dots, m\}$ such that $\sum_{i\in I} a_i = \sum_{i\in J-I} a_i$. Without loss of generality, we further suppose that $|I| \geq |J - I|$. For each site s_i with $i \in I$, we choose to place its label at the first stack. The remaining labels are placed at the second stack. The leaders of the sites with labels at the first stack are of type-*o*. In this case, the ports of both sites and labels can be chosen arbitrarily. Since the labeling is *tight* (i.e. the sum of the label heights on each stack is equal to H), we use the fact that the labels are open and use the gaps between them as *corridors* to route the leaders, that connect sites with labels at the second stack. Since we assumed that $|I| \geq |J - I|$, there exist enough corridors to route all leaders: The leader which corresponds to the lowest label that has not been routed yet, can use the lowest available corridor. In this case the site ports are defined based on the corridors, whereas the label ports can be chosen arbitrarily again. □

4.2 Two Candidate Points with Type-*opo* Leaders at Two Stacks on One Side

We will show that the problem remains NP-hard even if we restrict ourselves in sites, which may have two candidate points, i.e. leader of site s_i connects either point $p_i^1 = (x_i^1, y_i^1)$ or point $p_i^2 = (x_i^2, y_i^2)$ with label l_i. To show NP-hardness, we reduce the following variant of Partition to our problem. The NP-hardness of this problem follows easily from the Even Odd Partition problem (see [9] pp. 223).

Lemma 5 (RPartition). *Given* $2m$ *non-negative integers* $a_1, a_2, \dots a_{2m}$*, the problem of finding a subset* I *of* $J = \{1, 2, \dots 2m\}$ *such that the following three conditions are satisfied is* NP*-hard.* 1) *I contains exactly one of* $\{2i-1, 2i\}$ *for* $i = 1, 2, \dots m$. 2) $\sum_{i\in I} a_i = \sum_{i\in J-I} a_i$ *and* 3) $\sum_{i\in I \& i\leq k} a_i < \sum_{i\in J-I \& i\leq k} a_i$ *for* $k = 2, 4, \dots 2m-2$.

Theorem 4. *Given a rectangle* R *of height* H*, a set* $P \subset R$ *of* n *sites, each associated with two candidate points, and a label of height* h_i *for each site* $s_i \in P$*, it is* NP*-hard to place all labels at two stacks on one side of* R *with non-intersecting type-opo leaders.*

Proof. Let $A = \{a_1, a_2, \dots a_{2m}\}$ be an instance of RPartition. We will construct an instance B of our problem as follows: Let C be a very large number, e.g. $C = (2m+1)^2 \sum_{i\in J} a_i$. Set $P = \{s_1, s_2, \dots, s_{2m}\}$ consists of $2m$ sites. Site s_i is associated with $p_i^1 = (x_i, y_i^1)$ and $p_i^2 = (x_i, y_i^2)$. Consecutive sites s_{2i-1} and s_{2i}, $i = 1, 2, \dots, m$, form m parallelograms r_i, $i = 1, 2, \dots, m$, such that $y_{2i-1}^1 < y_{2i}^1 < y_{2i-1}^2 < y_{2i}^2$ and $|y_{2i-1}^2 - y_{2i}^1| = \frac{a_{2i-1}+a_{2i}}{2} + 1$. We assume that parallelogram r_{i-1} is placed lower than r_i. The vertical distance between two consecutive parallelograms is C, whereas the vertical distance between the bottommost (topmost) parallelogram r_1 (r_m) and the bottommost (topmost respectively) side of the enclosing rectangle R is $C/2$. The height of the enclosing rectangle is $H = m(C+1) + \frac{1}{2}\sum_{i \in J} a_i$. The label l_i of site s_i has height $h_i = C + a_i + 1$, thus $\sum_{i \in J} h_i = 2m(C+1) + \sum_{i \in J} a_i$. Observe, that both stacks contribute $2H$ height, which is equal to the sum of all label heights.

One can see that the construction ensures that the same number of labels are placed at the two stacks and that all leaders should bend in the first track routing area. Two consecutive sites s_{2i-1} and s_{2i}, $i = 1, 2, \ldots m$ can not have their labels both at the same stack, because at least one corridor is lost and therefore at least one label at the second stack can not be routed. To avoid leader crossings, the order of indices should be preserved at both stacks, i.e. if $i < j$ then label l_i will be stacked lower than l_j. To connect all sites with their labels, it must either hold $\sum_{i \in I \& i \leq k} h_i < \sum_{i \in J-I \& i \leq k} h_i$ or $\sum_{i \in I \& i \leq k} h_i > \sum_{i \in J-I \& i \leq k} h_i$, for all $k = 2, 4, \ldots 2m-2$, which is equivalent to condition (3) of RPartition. The indices of the sites with labels at the first stack imply the partition I of J.

Suppose that we have a subset I of J of A such that all three conditions of RPartition are satisfied. If $i \in I$, then the label of site s_i is placed at the first stack preserving the order of indices. The remaining labels ($i \in J - I$) are placed at the second stack in the same manner. A legal labeling is obtained by taking the lowest site which has not been routed. If its label is to be placed at the second stack, use the lowest available corridor for its leader, else route it at the first stack with a type-o leader. For two consecutive sites s_{2i-1} and s_{2i} we can determine in constant time which points will be used, such that their leaders do not intersect. □

4.3 Type-*po* Leaders at Two Stacks on One Side

Following similar arguments as in proof of Theorem 4, one can show that the problem remains NP-hard if we use type-*po* leaders, even if we restrict ourselves to a point $p_i = (x_i, y_i)$ or two candidate points $p_i^1 = (x_i^1, y_i^1)$ and $p_i^2 = (x_i^2, y_i^2)$ for each site s_i. Recall that for the case of two candidate points the leader of each site s_i connects either point p_i^1 or point p_i^2 with label l_i. The corresponding theorems follow. Detailed proofs of these theorems are given in the full version of the paper (see [2]).

Theorem 5. *Given a rectangle R of height H, a set $P \subset R$ of n points and a label of height h_i for each site $s_i \in P$, it is NP-hard to place all labels at two stacks on one side of R with non-intersecting type-po leaders.*

Theorem 6. *Given a rectangle R of height H, a set $P \subset R$ of n sites, each associated with two candidate points, and a label of height h_i for each site, it is NP-hard to place all labels at two stacks on one side of R with non-intersecting type-po leaders.*

Since, each point site can be thought as a line site of zero length, Corollary 1 follows immediately from Theorem 5.

Corollary 1. *Given a rectangle R of height H, a set $P \subset R$ of n lines and a label of height h_i for each site $s_i \in P$, it is NP-hard to place all labels at two stacks on one side of R with non-intersecting type-po leaders.*

5 Open Problems and Future Work

We presented results for the label size maximization problem and for the legal label placement for the case of two and three stacks of labels on the same side of R. No results are known regarding the total leader length minimization and the minimization of the total number of bends. Another line of research is to design good approximation algorithms that solve the problems, that are proved to be NP-hard.

References

1. M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Boundary labelling of optimal total leader length. In *Proc. 10th Panhellenic Conference On Informatics (PCI2005), LNCS 3746*, pages 80–89, 2005.
2. M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Multi-stack boundary labeling problems, 2006. Technical report, Available online http://www.math.ntua.gr/aarg/.
3. M. A. Bekos, M. Kaufmann, K. Potika, and A. Symvonis. Polygon labelling of minimim leader length. In *Asia Pacific Symposium on Information Visualisation (APVIS2006), CRPIT 60*, pages 15–21, 2006.
4. M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. *Computational Geometry: Theory and Applications*. Available online http://www.sciencedirect.com/.
5. M. A. Bekos, M. Kaufmann, A. Symvonis, and A. Wolff. Boundary labeling: Models and efficient algorithms for rectangular maps. In *Proc. 12th Int. Symposium on Graph Drawing (GD'04), LNCS 3383*, pages 49–59, 2004.
6. M. A. Bekos and A. Symvonis. Bler: A boundary labeller for technical drawings. In *Proc. 13th Int. Symposium on Graph Drawing (GD'05), LNCS 3843*, pages 503–504, 2005.
7. P. Eades, T. Lin, and X. Lin. Minimum size h-v drawings. In *Advanced Visual Interfaces (Proceedings of AVI '92), volume 36 of World Scientific Series in Computer Science*, pages 386–394, 1992.
8. M. Formann and F. Wagner. A packing problem with applications to lettering of maps. In *Proc. 7th Annuual ACM Symposium on Computational Geometry (SoCG'91)*, pages 281–288, 1991.
9. M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, New York, NY, 1979, 1979.
10. SmartDraw-7. Product web site: http://www.smartdraw.com.
11. L. Stockmeyer. Optimal orientations of cells in slicing floorplan designs. *Information and Control*, 57(2–3):91–101, 1983.
12. A. Wolff and T. Strijk. The Map-Labeling Bibliography. http://i11www.ira.uka.de/map-labeling/bibliography, 1996.

Computing a Center-Transversal Line[⋆]

Pankaj K. Agarwal[1], Sergio Cabello[2], J. Antoni Sellarès[3], and Micha Sharir[4]

[1] Department of Computer Science, Duke University, USA
pankaj@cs.duke.edu
[2] Dep. of Mathematics, IMFM and FMF, University of Ljubljana, Slovenia
sergio.cabello@fmf.uni-lj.si
[3] Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain
sellares@ima.udg.es
[4] School of Computer Science, Tel Aviv University, Israel, and Courant Institute of Mathematical Sciences, New York University, USA
michas@post.tau.ac.il

Abstract. A center-transversal line for two finite point sets in $\mathbb{R}^3$ is a line with the property that any closed halfspace that contains it also contains at least one third of each point set. It is known that a center-transversal line always exists [12,24], but the best known algorithm for finding such a line takes roughly n^{12} time. We propose an algorithm that finds a center-transversal line in $O(n^{1+\varepsilon}\kappa^2(n))$ worst-case time, for any $\varepsilon > 0$, where $\kappa(n)$ is the maximum complexity of a single level in an arrangement of n planes in $\mathbb{R}^3$. With the current best upper bound $\kappa(n) = O(n^{5/2})$ of [21], the running time is $O(n^{6+\varepsilon})$, for any $\varepsilon > 0$. We also extend the concept of center-transversal line to that of bichromatic depth of lines in space, and give an algorithm that computes a deepest line exactly in time $O(n^{1+\varepsilon}\kappa^2(n))$, and a linear-time approximation algorithm that computes, for any specified $\delta > 0$, a line whose depth is at least $1-\delta$ times the maximum depth.

1 Introduction

Two classical notions in discrete geometry are the notions of center points and ham-sandwich cuts. Given a set P of points in $\mathbb{R}^d$, a point q, not necessarily in P, is a *center point* with respect to P if any closed halfspace that contains q also contains at least $|P|/(d+1)$ points of P. The existence of center points

[⋆] P.A. was supported by NSF under grants CCR-00-86013 EIA-98-70724, EIA-99-72879, EIA-01-31905, and CCR-02-04118. S.C. was partially supported by the European Community Sixth Framework Programme under a Marie Curie Intra-European Fellowship, and by the Slovenian Research Agency, project J1-7218-0101. J.A.S. was partially supported by grant TIN2004-08065-C02-02 of the Spanish Ministry of Education and Science (MEC). M.S. was partially supported by NSF Grants CCR-00-98246 and CCF-05-14079, by grant 155/05 of the Israel Science Fund, and by the Hermann Minkowski–MINERVA Center for Geometry at Tel Aviv University. P.A. and M.S. were also supported by a joint grant from the U.S.-Israeli Binational Science Foundation.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 93–104, 2006.

is a consequence of Helly's theorem [15]. Given d finite point sets $P_0, \dots, P_{d-1}$ in $\mathbb{R}^d$ with n points in total, a *ham-sandwich cut* is a hyperplane h such that each of the open halfspaces bounded by h contains at most $|P_i|/2$ points of P_i, for every $i = 0, 1, \dots, d-1$. Dol'nikov [12], and Živaljević and Vrećica [24] proved the following theorem, called *center-transversal theorem*, which yields a generalization of center points and ham-sandwich cuts.

Theorem 1 (Center-Transversal Theorem). *Given $k+1$ finite point sets $P_0, P_1, \dots, P_k$ in $\mathbb{R}^d$, for any $0 \le k \le d-1$, there exists a k-flat f such that any closed halfspace that contains f also contains at least $\frac{1}{d-k+1}|P_i|$ points of P_i, for each $i = 0, 1, \dots, k$.*

Observe that when $k = 0$, f is a center point, and when $k = d-1$, f is a ham-sandwich cut. Therefore, the center-transversal theorem can be seen as an "interpolation" between these two theorems. A weaker result with $|P_i|/(d+1)$ instead of $|P_i|/(d-k+1)$ can easily be obtained by considering the k-flat passing through a center point of each of the P_i, $i = 0, 1, \dots, k$.

In this paper we consider in detail the case $d = 3$, $k = 1$. Given two finite point sets P_0, P_1 in $\mathbb{R}^3$, we say that a line ℓ is a *center-transversal line* for P_0, P_1 if any closed half-space that contains ℓ also contains at least $|P_i|/3$ points of P_i, for $i = 0, 1$. The center-transversal theorem asserts that, for any finite point sets P_0, P_1 in $\mathbb{R}^3$, there exists a center-transversal line. However, the original proofs [12,24] of this result are non-constructive and do not lead to an algorithm for finding a center-transversal line. The running time of the best known algorithm for this problem [5] is rather large (about n^{12}—see below). We present a considerably more efficient algorithm for finding such a line, and consider several other related problems.

Related work. A more detailed review of center points, ham sandwich cuts, and related problems can be found in Matoušek [15]. Efficient algorithms are known for computing a center point in $\mathbb{R}^2$ and $\mathbb{R}^3$ [14,16]. A center point in $\mathbb{R}^d$ can be found using linear programming with $\Theta(n^d)$ linear inequalities, and there exists a faster algorithm, due to Clarkson et al. [10], for computing an *approximate* center point in arbitrary dimensions; that is, a point q such that any closed halfspace containing q contains at least $\Omega(n/d^2)$ points of P. Efficient algorithms have also been developed for constructing the *center region*, namely, the set of all center points, in $\mathbb{R}^2$ and $\mathbb{R}^3$ [4,6]. The concept of center point leads to generalizations that have been useful in robust statistics. The *halfspace depth* (also called location depth, data depth) of a point q relative to a data set P in $\mathbb{R}^d$, is the smallest number of data points in any closed halfspace whose boundary passes through q. A center point is a point with depth at least $|P|/(d+1)$, and a halfspace median, or a *Tukey point*, is a point with maximum halfspace depth. Chan [6], improving upon previous results, has recently obtained a randomized $O(n \log n + n^{d-1})$ expected-time algorithm for computing a Tukey median point in $\mathbb{R}^d$.

The problem that we consider can be related to *multivariate regression depth*, a generalization, introduced by Bern and Eppstein [5], of *regression depth*, a quality measure for robust linear regression defined by Rousseeuw and Hubert [19,20]. In particular, Bern and Eppstein [5] give a general-purpose algorithm, which can be easily modified to yield an algorithm that constructs a center-transversal line in $\mathbb{R}^3$ in $O(n^{12+\varepsilon})$ time, for any $\varepsilon > 0$.

Our contributions. Let P_0, P_1 be two finite point sets in $\mathbb{R}^3$ with a total of n points.

- We present an algorithm that constructs a center-transversal line for P_0 and P_1 in $O(n^{1+\varepsilon}\kappa^2(n))$ worst-case time, for any $\varepsilon > 0$, where $\kappa(n)$ is the maximum complexity of a single level in an arrangement of n planes in $\mathbb{R}^3$. With the current best upper bound $\kappa(n) = O(n^{5/2})$ of [21], the running time is $O(n^{6+\varepsilon})$, for any $\varepsilon > 0$. See Section 2. This is a considerable improvement over the algorithm by Bern and Eppstein [5].[1]
- We describe a randomized algorithm that, for a given direction u, in $O(n \log n)$ expecetd time whether there exists a center-transversal line of P_0 and P_1 i in direction u. Because of lack of space, we omit this algorithm from this extended abstract.
- We introduce the notion of the *bichromatic depth* of a line ℓ, with respect to P_0 and P_1, extending similar earlier concepts. Specifically, it is the minimum fraction size ρ of the points in either set that lie in a halfspace that contains ℓ; that is, each halfspace containing ℓ contains at least $\rho|P_0|$ points of P_0 and $\rho|P_1|$ points of P_1. This concept generalizes that of center-transversal line (which has bichromatic depth at least 1/3). We show how to compute a deepest line in $O(n^{1+\varepsilon}\kappa^2(n))$ time, for any $\varepsilon > 0$, and give a linear-time approximation algorithm that computes, for any $\delta > 0$, a line whose depth is at least $1 - \delta$ times the maximum depth. See Section 3.

2 Finding a Center-Transversal Line

We consider the problem of computing a center-transversal line in dual space, where the problem is reformulated in terms of levels in arrangements of planes. We generate a set of candidate lines that is guaranteed to contain a center-transversal line and use a data structure to determine which of these candidate lines is a center transversal line. For simplicity, we assume that $|P_0|$ and $|P_1|$ are multiples of 3, and that the points $P_0 \cup P_1$ are in general position in the sense that no four of them are coplanar.

Center-transversal lines in the dual. The widely used *duality* transform maps a point p in $\mathbb{R}^d$ to a hyperplane p^* in $\mathbb{R}^d$ and vice-versa, so that the incidence and above/below relationships are preserved. There are many variants of duality [15]; we use the following one: A point $a = (a_1, \ldots, a_d) \in \mathbb{R}^d$ is mapped to the nonvertical hyperplane $a^* : x_d = a_1x_1 + \cdots + a_{d-1}x_{d-1} - a_d$, and a hyperplane $h : x_d = \alpha_1x_1 + \cdots + \alpha_{d-1}x_{d-1} + \alpha_d$ is mapped to the point $h^* = (\alpha_1, \ldots, \alpha_{d-1}, -\alpha_d)$,

[1] We note though that an algorithm with running time near n^8 is not hard to obtain.

so $(a^*)^* = a$. A point p lies below (resp., above, on) a hyperplane h if the dual point h^* lies below (resp., above, on) the dual hyperplane p^*. The *pencil* of hyperplanes passing through a line ℓ in $\mathbb{R}^d$, for $d \geq 3$, maps to the set of points in $\mathbb{R}^d$ lying on a line ℓ^*; we refer to ℓ^* as the dual of ℓ. For a set A of objects, set $A^* = \{a^* \mid a \in A\}$.

Let P be a set of n points in $\mathbb{R}^3$, and let $H = P^*$ be the set of n non-vertical planes in $\mathbb{R}^3$ dual to the points in P. The *level* of a point $p \in \mathbb{R}^3$, with respect to H, is the number of planes in H that lie *below* p. For $0 \leq k < n$, the *k-level* of H, denoted $\mathcal{L}_k(H)$ (or simply $\mathcal{L}_k$ if the set H is understood), is the closure of the set of all points on any of the planes of H that are at level k. The k-level $\mathcal{L}_k$ is a *polyhedral terrain*, that is, an xy-monotone piecewise-linear continuous surface formed by a subset of the faces of the arrangement $\mathcal{A}(H)$. The combinatorial complexity of $\mathcal{L}_k$ is the number of faces of all dimensions in $\mathcal{L}_k$. Let $\kappa(n)$ denote the maximum complexity of a level in any arrangement of n planes in $\mathbb{R}^3$. The best known upper bound for $\kappa(n)$ is $O(n^{5/2})$ [21], which differs substantially from the best known lower bound $n^2 e^{\Omega(\sqrt{\log n})}$ [23]. See [3] for more details on arrangements and levels.

If h is a plane in $\mathbb{R}^3$ so that each of the two halfspaces bounded by h contains at least k points of P, then h^* lies between $\mathcal{L}_k(H)$ and $\mathcal{L}_{n-k}(H)$. If ℓ is a line in $\mathbb{R}^3$ so that any halfspace containing ℓ contains at least k points of P, then the entire dual line ℓ^* lies between $\mathcal{L}_k(H)$ and $\mathcal{L}_{n-k}(H)$. Hence, the problem of computing a center-transversal line for P_0 and P_1 reduces to computing a line in the dual space that lies above $\Sigma_0 = \mathcal{L}_{n_0/3}(H_0), \Sigma_1 = \mathcal{L}_{n_1/3}(H_1)$ and below $\Sigma_2 = \mathcal{L}_{2n_0/3}(H_0), \Sigma_3 = \mathcal{L}_{2n_1/3}(H_1)$, where $H_i = P_i^*$ and $n_i = |P_i|$ for $i = 0, 1$. We note that each of these four terrains can be computed in $O(n^\varepsilon \kappa(n))$ time, for any $\varepsilon > 0$ [2].

We thus have four terrains $\Sigma_0, \Sigma_1, \Sigma_2, \Sigma_3$, and we wish to compute a line that lies above Σ_0, Σ_1 and below Σ_2, Σ_3. Note that such a line cannot be *z-vertical*, i.e., parallel to the z-axis. Let E_i be the set of edges in Σ_i, for $i = 0, 1, 2, 3$, and $E = \bigcup_{i=0}^{3} E_i$. Set $m := |E| \leq 4\kappa(n)$, and assume that $m \geq n$ (or else the problem can be solved much faster than the time bound of our algorithm). Let $H = H_0 \cup H_1$. Each edge in E_i lies in the intersection line of a pair of planes in H. We define a "sidedness function" $\chi : E \to \{+1, -1\}$, where $\chi(e) = +1$ if $e \in E_0 \cup E_1$ and $\chi(e) = -1$ if $e \in E_2 \cup E_3$. Let V be the set of endpoints of edges in E. By the general-position assumption, each point of V is incident upon at most three edges of E. For an object (point, line, segment) Δ in $\mathbb{R}^3$, let $\tilde{\Delta}$ denote its xy-projection in $\mathbb{R}^2$.

Definition 1. *Let ℓ be a nonvertical line in $\mathbb{R}^3$, and let e be a nonvertical segment in $\mathbb{R}^3$ so that $\tilde{\ell}$ intersects $\tilde{e}$. We say that ℓ lies* above *(resp.,* below*) e if the oriented line in the $(+z)$-direction that passes through $\tilde{\ell} \cap \tilde{e}$ meets e before (resp., after) ℓ. The line ℓ is in* compliance *with an edge $e \in E$ if (i) $\tilde{\ell}$ does not intersect $\tilde{e}$, or (ii) ℓ does not lie below (resp., above) e if $\chi(e) = +1$ (resp., $\chi(e) = -1$). We say that ℓ is in compliance with a subset $R \subseteq E$ if it is in compliance with every edge in R. In particular, we have:*

Lemma 1. *A nonvertical line ℓ in $\mathbb{R}^3$ lies above Σ_0, Σ_1 and below Σ_2, Σ_3 if and only if ℓ is in compliance with E.*

The problem of computing a center-transversal line now reduces to finding a line that is in compliance with E. Let $\mathbb{L}$ be the set of all lines in $\mathbb{R}^3$ that are not parallel to the yz-plane. We restrict the search for a line that is in compliance with E to lines in $\mathbb{L}$. This involves no loss of generality: The lines in $\mathbb{R}^3$ parallel to the yz-plane have three degrees of freedom and a center-transversal line among them, if there exists one, can be found using a much simpler (and more efficient) algorithm. Alternatively, we can run our algorithm twice, exchanging the roles of the x- and y-axes in the second run.

Overview of the algorithm. We show that, for each line $\ell \in \mathbb{L}$, there exists a "witness set" of $O(n)$ edges of E, so that ℓ is in compliance with E if and only if it is in compliance with its witness set. We then group the lines in $\mathbb{L}$ into equivalence classes so that all lines in the same class have the same witness set. Using this reduction, we present an algorithm that works in three stages. The first stage, called the *filtering stage*, splits the problem into $O(m^2/n^2)$ subproblems, each aiming to compute a line that is in compliance with some set of $O(n)$ edges. The second stage, a *recursive candidate generation stage*, computes, for each subproblem, a set of $O(n^{3+\varepsilon})$ candidate lines, for any $\varepsilon > 0$, which is guaranteed to contain a line in compliance with the corresponding subset if there exists one. The final stage, the *verification stage*, checks which of the candidate lines generated by the previous step is in compliance with E, and report the first such line that it encounters (which is guaranteed to exist). We now describe each of these steps in detail.

Witness sets and equivalence classes. For a line $\ell \in \mathbb{L}$ and a subset $R \subseteq E$ of edges, we define the *witness set* of ℓ for R, denoted by $W(\ell, R)$, as follows. For $i = 0, 1, 2, 3$, let $R_i \subseteq R$ be the sequence of edges in $R \cap E_i$ whose xy-projections intersect $\tilde{\ell}$, sorted by the order of the intersection points along $\tilde{\ell}$. For a plane $h \in H_0 \cup H_1$, let $e^-_{h,i}, e^+_{h,i} \in R_i$ be, respectively, the first and the last edges in the i-th sequence that lie on h, where only planes in H_0 (resp., H_1) are considered for $i = 0, 2$ (resp., $i = 1, 3$). We set

$$W(\ell, R) = \{e^-_{h,i}, e^+_{h,i} \mid h \in H,\ 0 \leq i \leq 3\}.$$

By definition, $\tilde{\ell}$ intersects the xy-projection of every edge in $W(\ell, R)$; $|W(\ell, R)| = O(n)$.

Lemma 2. *For a subset $R \subseteq E$, a line $\ell \in \mathbb{L}$ is in compliance with R if and only if ℓ is in compliance with $W(\ell, R)$.*

The proof of the lemma follows from the simple observation that if ℓ lies above (resp., below) both $e^-_{h,i}, e^+_{h,i}$ then it lies above (resp., below) all edges in R_i that lie in h.

We define, for a subset $R \subseteq E$, an equivalence relation on $\mathbb{L}$ so that for any two lines ℓ_1, ℓ_2 in the same equivalence class, $W(\ell_1, R) = W(\ell_2, R)$. This will discretize the search for a center-transversal line. For this we need a few notations. For a point or a line ξ in $\mathbb{R}^3$, let $\varphi(\xi)$ denote the dual (in $\mathbb{R}^2$) of $\tilde{\xi}$, i.e., $\varphi(\xi) = \tilde{\xi}^*$. [2] For an edge $e = uv$ in E, let $\varphi(e) \subseteq \mathbb{R}^2$ be the double wedge that is formed by the lines $\varphi(u)$ and $\varphi(v)$ and does not contain the line in $\mathbb{R}^2$ passing through their intersection point and parallel to the y-axis. By standard properties of the duality transform in $\mathbb{R}^2$, a line γ in $\mathbb{R}^2$ intersects $\tilde{e}$ if and only if $\gamma^* \in \varphi(e)$. Moreover if the points $\gamma_1^*, \gamma_2^* \in \mathbb{R}^2$ lie in the same (left or right) wedge of $\varphi(e)$, then γ_1, γ_2 intersect $\tilde{e}$ from the *same side*, in the sense that the same endpoint of $\tilde{e}$ lies in each of the positive halfplanes bounded by γ_1 and γ_2, respectively (that is, the halfplanes above these lines).

Let $R \subseteq E$ be a fixed subset of edges, let $V_R \subseteq V$ be the set of endpoints of the edges in R, and let $\Lambda(R) = \{\varphi(v) \mid v \in V_R\}$ be the corresponding set of lines in $\mathbb{R}^2$. For each face f in the arrangement $\mathcal{A}(\Lambda(R))$ of $\Lambda(R)$, let $R(f)$ denote the set of those edges $e \in R$ for which $\varphi(e)$ contains f. For a line $\ell \in \mathbb{L}$, if f is the face containing $\varphi(\ell)$ then, by construction, $R(f)$ is the set of edges of R whose xy-projections intersect $\tilde{\ell}$. By definition, $W(\ell, R) \subseteq R(f)$.

Definition 2. *We call two lines* $\ell_1, \ell_2 \in \mathbb{L}$ equivalent *(with respect to R), denoted by* $\ell_1 \equiv_R \ell_2$*, if* $\varphi(\ell_1)$ *and* $\varphi(\ell_2)$ *lie in the same face of* $\mathcal{A}(\Lambda(R))$.

Lemma 3. *Let* $R \subseteq E$ *be a set of edges, and let* $\ell_1, \ell_2 \in \mathbb{L}$ *be two lines so that* $\ell_1 \equiv_R \ell_2$*. Then* $W(\ell_1, R) = W(\ell_2, R)$.

Proof. Let f be the face of $\mathcal{A}(\Lambda(R))$ that contains $\varphi(\ell_1)$ and $\varphi(\ell_2)$. Set $R_i(f) := R(f) \cap E_i$ and $L_i := \Lambda(R_i(f)) \subseteq \Lambda(R)$, for $i = 0, 1, 2, 3$. Clearly, $\varphi(\ell_1), \varphi(\ell_2)$ lie in the same face of $\mathcal{A}(L_i)$. Since the edges of E_i all belong to the same terrain, their xy-projections are pairwise disjoint. An easy observation (due to [1]) shows that $\tilde{\ell}_1, \tilde{\ell}_2$ intersect the xy-projections of the edges in $R_i(f)$ in the same order. This immediately implies that $W(\ell_1, R) \cap E_i = W(\ell_2, R) \cap E_i$, from which the lemma follows.

In view of the preceding lemma, we define, for each face f of $\mathcal{A}(R)$, $W_f(R) \subseteq R$ to be the common witness set for any line in the equivalence class corresponding to f.

The filtering stage. Given a set L of lines in $\mathbb{R}^2$, a triangle Δ_0, and a parameter $1 \le r \le |L|$, a $(1/r)$-*cutting* of (L, Δ_0) is a triangulation Ξ of Δ_0 so that each triangle of Ξ is crossed by at most $|L|/r$ lines of L. It is known that a $(1/r)$-cutting consisting of $O(r^2)$ triangles, along with the set of lines crossing each of its triangles, can be computed in $O(|L|r)$ time [7].

Let $\Lambda = \Lambda(E)$. We set $\Delta_0 = \mathbb{R}^2$ and $r = m/n$, and compute a $(1/r)$-cutting Ξ of (Λ, Δ_0). For each triangle $\Delta \in \Xi$, let Λ_Δ be the set of lines of Λ that cross Δ; since Ξ is a $(1/r)$-cutting, we have $|\Lambda_\Delta| \le m/r = n$. Let $E_\Delta \subseteq E$ be the set of edges $e = uv$ so that either $\varphi(u) \in \Lambda_\Delta$ or $\varphi(v) \in \Lambda_\Delta$. Since each vertex of

[2] Note that $\varphi(\ell)$ is not defined if ℓ is parallel to the yz-plane. That is why we exclude these lines from $\mathbb{L}$.

V is an endpoint of at most three edges of E, we have $|E_\Delta| \leq 3|\Lambda_\Delta| \leq 3n$. For each $\Delta \in \Xi$, let $F_\Delta = \{e \in E \setminus E_\Delta \mid \Delta \subseteq \varphi(e)\}$. We refer to the edges in E_Δ as *short* and to the edges in F_Δ as *long*. Finally, let $\mathbb{L}_\Delta = \{\ell \in \mathbb{L} \mid \varphi(\ell) \in \Delta\}$.

Since Δ is contained in a face of $\mathcal{A}(\Lambda(F_\Delta))$ (the arrangement of lines dual to the xy-projections of the endpoints of the edges in F_Δ), Lemma 3 implies that $W(\ell, F_\Delta)$ is the same for all lines $\ell \in \mathbb{L}_\Delta$; let W_Δ denote this common witness set. Observe that $|W_\Delta| = O(n)$.

If two triangles Δ and Δ' in Ξ share an edge, then $F_\Delta \oplus F_{\Delta'} \subseteq E_\Delta \cup E_{\Delta'}$. Therefore W_Δ can be computed from $W_{\Delta'}$ in $O(|E_\Delta| + |E_{\Delta'}|) = O(n)$ time. Hence, by performing a traversal of Ξ, we can compute W_Δ for all triangles $\Delta \in \Xi$, in overall time $O(m^2/n)$.

The next lemma follows from Lemmas 2 and 3.

Lemma 4. *For any $\Delta \in \Xi$, a line $\ell \in \mathbb{L}_\Delta$ is in compliance with E if and only if ℓ is in compliance with $E_\Delta \cup W_\Delta$.*

Hence, for each $\Delta \in \Xi$, we have a subproblem $(\Delta, E_\Delta, W_\Delta)$, in which we want to determine whether there is a line in $\mathbb{L}_\Delta$ that is in compliance with $E_\Delta \cup W_\Delta$ (and thus with E). Since $\bigcup_\Delta \mathbb{L}_\Delta = \mathbb{L}$, these subproblems together exhaust the overall problem of computing a line in $\mathbb{L}$ that is in compliance with E. There are $O(m^2/n^2)$ such subproblems, and the total time spent in generating them is $O(m^2/n)$.

The recursive candidate generation stage. Let $(\Delta, E_\Delta, W_\Delta)$ be one of the subproblems generated in the previous stage. We generate a set of "candidate" lines that contains a line in compliance with $E_\Delta \cup W_\Delta$ if there exists one. Let $\ell \in \mathbb{L}_\Delta$ be such a line. We move it around while keeping it in the set $\mathbb{L}_\Delta$ and in compliance with $E_\Delta \cup W_\Delta$, until we reach a critical position of ℓ at which one of the following events occurs (for the following enumeration, recall that passing above, below, or through an endpoint of an edge in W_Δ can occur only when $\varphi(\ell)$ reaches the boundary of Δ):

(E0) $\varphi(\ell)$ is a vertex of Δ;
(E1) ℓ passes through a pair of endpoints of edges in E_Δ;
(E2) ℓ passes through an endpoint of an edge in E_Δ, $\varphi(\ell)$ lies on an edge of Δ, and ℓ touches the relative interior of an edge of $E_\Delta \cup W_\Delta$;
(E3) ℓ passes through an endpoint of an edge in E_Δ and touches the relative interior of two edges of $E_\Delta \cup W_\Delta$;
(E4) $\varphi(\ell)$ lies on an edge of Δ, and ℓ touches the relative interior of three edges of $E_\Delta \cup W_\Delta$;
(E5) ℓ touches the relative interior of four edges of $E_\Delta \cup W_\Delta$.

Since (E0)–(E4) are defined by at most three edges of $E_\Delta \cup W_\Delta$ and there are $O(1)$ lines for each such event (assuming general position), we generate all critical lines of these types (the $O(n^3)$ cost of producing these lines is subsumed by the cost of generating the lines of type (E5)—see below). We add all the resulting lines that belong to $\mathbb{L}_\Delta$ to the candidate set. Hence, it suffices to describe an algorithm for computing the set of candidate lines that satisfy (E5).

Let $\mathcal{C}(\Delta, E_\Delta, W_\Delta)$ denote this set. We compute a superset of $\mathcal{C}(\Delta, E_\Delta, W_\Delta)$ with a divide-and-conquer algorithm that employs *Plücker coordinates* [18]. Our approach for generating candidate lines is very similar to that used by Pellegrini [17].

Before describing the algorithm, we briefly review the representation of lines in Plücker space. An oriented line ℓ in $\mathbb{R}^3$ can be mapped to a point $\pi(\ell) \in \mathbb{R}^5$, called the *Plücker point* of ℓ, that lies on the so-called 4-dimensional *Plücker hypersurface Π*, or to a hyperplane $\varpi(\ell)$ in $\mathbb{R}^5$, called the *Plücker hyperplane* of ℓ. (The actual Plücker space is the *real projective* 5-space, but since we exclude lines parallel to the yz-plane, it is easy (though some care is needed) to embed the Plücker structure into the real 5-dimensional space.) Abusing the notation a little, we use $\pi(e)$ and $\varpi(e)$ to denote the Plücker point and hyperplane, respectively, of the line supporting an oriented segment e in $\mathbb{R}^3$.

We orient every line of $\mathbb{L}$ and every edge of E in the $(+x)$-direction (this is well defined for lines in $\mathbb{L}$, by definition, and for edges of E, by the general position assumption). For two oriented lines ℓ_1, ℓ_2 in $\mathbb{R}^3$, $\pi(\ell_1)$ lies above $\varpi(\ell_2)$ (which is the same as $\pi(\ell_2)$ lying above $\varpi(\ell_1)$) if and only if the simplex spanned by a vector $\boldsymbol{u}_1$ lying on ℓ_1 with the same orientation, and by a vector $\boldsymbol{u}_2$ lying on ℓ_2 with the same orientation, is positively oriented. This is easily seen to imply that, when ℓ_1 and ℓ_2 are non-vertical, ℓ_1 passes above ℓ_2 if and only if either (i) $\pi(\ell_1)$ lies above $\varpi(\ell_2)$ and $\tilde{\ell_1}$ lies counterclockwise to $\tilde{\ell_2}$, or (ii) $\pi(\ell_1)$ lies below $\varpi(\ell_2)$ and $\tilde{\ell_1}$ lies clockwise to $\tilde{\ell_2}$; see [18] for more details.

We now proceed to describe the construction of the set of lines $\mathcal{C}(\Delta, E_\Delta, W_\Delta)$. We choose a constant r and construct a $(1/6r)$-cutting T of $(\Lambda(E_\Delta), \Delta)$. As in the filtering stage, we define, for each $\tau \in T$, $E_\tau \subseteq E_\Delta$ to be the set of short edges in τ, and $F_\tau \subseteq E_\Delta$ to be the set of long edges in τ. We have $|E_\tau| \leq 3|\Lambda(E_\Delta)|/6r \leq |E_\Delta|/r$. Set $W_\tau := F_\tau \cup W_\Delta$. Define $\mathbb{L}_\tau = \{\ell \in \mathbb{L}_\Delta \mid \varphi(\ell) \in \tau\}$, and note that $\bigcup_{\tau \in T} \mathbb{L}_\tau = \mathbb{L}_\Delta$. For each $\tau \in T$, we compute a set of *candidate* lines $\mathcal{C}_\tau \subset \mathbb{L}_\tau$, with the property that $\mathcal{C}(\Delta, E_\Delta, W_\Delta) \subseteq \bigcup_{\tau \in T} \mathcal{C}_\tau$.

Consider a triangle $\tau \in T$. We want to construct a set of candidate lines $\mathcal{C}_\tau$ that includes the lines in $\mathbb{L}_\tau$ of type (E5). Hence, it suffices to consider only the edges $E_\tau \cup W_\tau$ in its construction. The line ℓ is in compliance with an edge $e \in W_\tau$ if $\pi(\ell)$ lies in one specific halfspace Γ_e bounded by $\varpi(e)$. Γ_e depends on the function $\chi(e)$ and on the clockwise order of $\tilde{\ell}$ and $\tilde{e}$ (when oriented in the positive x-direction). Since τ is a subset of a fixed wedge of $\varphi(e)$, this clockwise order is the same for all lines $\ell \in \mathbb{L}_\tau$; hence Γ_e is the same halfspace for all lines in $\mathbb{L}_\tau$. Set $\mathcal{K} := \bigcap_{e \in W_\tau} \Gamma_e$; $\mathcal{K}$ is a convex polyhedron in $\mathbb{R}^5$ with $O(n)$ facets, so its overall combinatorial complexity is $O(n^2)$.

Let $\ell \in \mathbb{L}_\tau$ be a line that touches the relative interior of four edges of $E_\tau \cup W_\tau$, and let $B(\ell)$ denote the set of these four edges. There are four cases, depending on how many edges of W_τ the line ℓ touches.

$B(\ell) \subseteq W_\tau$. If all edges of $B(\ell)$ belong to W_τ and ℓ is in compliance with $E_\tau \cup W_\tau$, then $\pi(\ell) \in \mathcal{K}$. Since ℓ touches four edges of W_τ, it lies on an edge of $\mathcal{K}$. Therefore, we find lines of this type by intersecting each edge of $\mathcal{K}$ with the (quadratic) Plücker hypersurface Π, and by adding the (at most) two lines corresponding to the two intersection points to the candidate set $\mathcal{C}_\tau$, if they belong to $\mathbb{L}_\tau$. The total time spent is $O(n^2)$.

$|B(\ell) \cap W_\tau| = 3$. For any line ℓ with this kind of contacts, $\pi(\ell)$ lies on the intersection edge of some 2-face of $\mathcal{K}$ and the Plücker hyperplane $\varpi(e)$ for some $e \in E_\tau$. For each pair $e \in E_\tau$ and 2-face ϕ of $\mathcal{K}$, we compute the at most two intersection points of $\phi \cap \varpi(e) \cap \varPi$, and add the corresponding lines to the candidate set $\mathcal{C}_\tau$, if they belong to $\mathbb{L}_\tau$. Since the polyhedron $\mathcal{K}$ has $O(n^2)$ 2-faces, the total number of lines generated in this case is $O(n^2|E_\tau|) = O(n^3/r)$, and their construction takes $O(n^3/r)$ time.

$|B(\ell) \cap W_\tau| = 2$. Let $e_1, e_2 \in E_\tau$ be the two edges that belong to $B(\ell)$. The Plücker subspace F of lines (in $\mathbb{L}$) that touch e_1 and e_2 is a 3-dimensional flat in $\mathbb{R}^5$, and $\pi(\ell) \in F \cap \mathcal{K}$. Since $F \cap \mathcal{K}$ is a convex 3-polyhedron with $O(n)$ facets, it only has $O(n)$ edges. We form, as above, the intersections of each edge of $F \cap \mathcal{K}$ with the Plücker surface $\varPi$, and add the (at most two) resulting lines to our candidate set $\mathcal{C}_\tau$, if they belong to $\mathbb{L}_\tau$. The total number of lines generated in this case is $O(|E_\tau|^2 n) = O(n^3/r^2)$, and their computation takes $O(|E_\tau|^2 n \log n) = O((n^3/r^2) \log n)$ time, where the costliest step is the construction, repeated $O(|E_\tau|^2)$ times, of convex 3-polyhedra, each defined by at most n inequalities.

$|B(\ell) \cap W_\tau| \leq 1$. We partition W_τ into $u = O(r)$ subsets $W_\tau^{(1)}, \ldots, W_\tau^{(u)}$ so that $|W_\tau^{(i)}| \leq n/r$ for each i. We recursively compute the set of candidate lines $\mathcal{C}(\tau, E_\tau, W_\tau^{(i)})$, for $1 \leq i \leq u$ and for $\tau \in T$. We thus recursively solve $O(r)$ subproblems, all of whose outputs are added to our candidate set $\mathcal{C}_\tau$. Clearly, all lines of this type (and perhaps more) are found by this recursive procedure.

The correctness of the procedure is fairly straightforward. Let $T(n)$ denote the maximum time needed to compute $\bigcup_{\tau \in T} \mathcal{C}_\tau$, which is a superset of $\mathcal{C}(\varDelta, E_\varDelta, W_\varDelta)$, when $|E_\varDelta|, |W_\varDelta| \leq n$. For each $\tau \in T$, we spend $O(n^2 + n^3/r + (n^3/r^2) \log n)$ time plus the time needed to solve $O(r)$ recursive calls where the size of each of the two sets of edges is at most n/r. Since the cutting T consists of $O(r^2)$ triangles, we obtain the following recurrence.

$$T(n) = O(r^3) T(n/r) + O(n^2 r^2 + n^3 r + n^3 \log n).$$

The solution of this recurrence is $T(n) = O(n^{3+\varepsilon})$, for any $\varepsilon > 0$ (for which we need to choose r sufficiently large, as a function of ε). The size of $\mathcal{C}(\varDelta, E_\varDelta, W_\varDelta)$ is also bounded by this quantity.

Repeating this procedure for the $O(m^2/n^2)$ subproblems generated by the filtering stage, we construct, in $O(m^2 n^{1+\varepsilon})$ overall time, a candidate set $\mathcal{C}$ of $O(m^2 n^{1+\varepsilon})$ lines.

The verification stage. To complete the algorithm, we test which of the lines in $\mathcal{C}$ is in compliance with E. Using the data structure described in [9], we can preprocess, in $O(m^{2+\varepsilon})$ time, each E_i into a data structure of size $O(m^{2+\varepsilon})$ so that we can determine in $O(\log n)$ time whether a line $\ell \in \mathbb{L}$ passes above or below the terrain $\varSigma_i$, or, equivalently, whether ℓ is in compliance with E_i.

Querying each line in $\mathcal{C}$ with this data structure for every E_i, we can determine, in $O(m^{2+\varepsilon} + m^2 n^{1+\varepsilon}) = O(m^2 n^{1+\varepsilon})$ time, which of the lines in $\mathcal{C}$ are in compliance with E. Since a center-transversal line always exists, it belongs to $\mathcal{C}$, by construction, and will be found by this procedure. Putting everything together, and recalling that $m \leq 4\kappa(n)$, where $\kappa(n)$ is the maximum complexity of a level in an arrangement of n planes in $\mathbb{R}^3$, we obtain the following main result of the paper. For the concrete time bound, we use the currently best known upper bound $\kappa(n) = O(n^{5/2})$ of [21].

Theorem 2. *A center-transversal line for two sets P_0, P_1 with a total of n points in $\mathbb{R}^3$ can be constructed in $O(n^{1+\varepsilon}\kappa^2(n))$ time, for any $\varepsilon > 0$. This time bound is $O(n^{6+\varepsilon})$, for any $\varepsilon > 0$.*

Terrains with many coplanar faces. Pellegrini [17] and Halperin and Sharir [13] have shown that the complexity of the envelope of lines above a terrain of complexity k is $O(k^{3+\varepsilon})$, for any $\varepsilon > 0$. The complexity of this envelope corresponds to the number of lines that are tangent to the terrain while lying above it. In our scenario, we have taken advantage of the fact that the faces of our terrains are contained in few planes. It is not clear how to plug this hypothesis into the techniques used in [13,17]. However, using the ideas of witness sets and the filtering stage as we have done, we directly obtain the following result, which may be of independent interest.

Theorem 3. *Let Σ be a terrain of complexity k in $\mathbb{R}^3$, all of whose facets lie on n different planes. Then the complexity of the envelope of lines that pass above Σ is $O(n^{1+\varepsilon}k^2)$, for any $\varepsilon > 0$.*

3 Variations

Bichromatically deepest line. The algorithm that we have presented in Section 2 can be extended so that, for any given number $\alpha \in [0,1]$, it finds a line ℓ with the property that any closed halfspace containing ℓ also contains at least $\lceil \alpha |P_i| \rceil$ points of P_i, for $i = 0,1$, or determines that no such line exists. The running time remains $O(n^{1+\varepsilon}\kappa^2(n))$, for any $\varepsilon > 0$.

We define the *bichromatic depth* of a line ℓ with respect to P_0, P_1 as follows:

$$\textsc{Depth}(\ell; P_0, P_1) = \min_h \left\{ \frac{|P_0 \cap h|}{|P_0|}, \frac{|P_1 \cap h|}{|P_1|} \right\} \in [0,1],$$

where the minimum is taken over all closed halfspaces h containing ℓ. Equivalently, $\textsc{Depth}(\ell; P_0, P_1) \geq \alpha$ means that any closed halfspace containing ℓ also contains at least $\lceil \alpha |P_i| \rceil$ points of P_i, for $i = 0,1$. A line ℓ_0 is a *bichromatically deepest line* if it has maximum bichromatic depth. The center-transversal theorem (Theorem 1) implies that there always exists a line of depth at least 1/3. By conducting a binary search and using the extended version of the algorithm of Section 2, we can easily find a line with maximum depth. We thus obtain the following.

Theorem 4. *Given two finite point sets P_0, P_1 in $\mathbb{R}^3$ with a total of n points, we can compute a bichromatically deepest line for P_0, P_1 in $O(n^{1+\varepsilon}\kappa^2(n))$ time, for any $\varepsilon > 0$.*

Computing an almost-deepest line. We next observe that, for any fixed $\delta > 0$, we can compute in linear time a line ℓ whose bichromatic depth with respect to P_0, P_1 is at least $1 - \delta$ times the maximum depth of a line. An *ε-approximation* of a point set P (with respect to closed halfspace ranges) is a subset $A \subseteq P$ such that, for any closed halfspace h we have

$$\left| \frac{|A \cap h|}{|A|} - \frac{|P \cap h|}{|P|} \right| \le \varepsilon.$$

As is well known [8], for any fixed ε, an ε-approximation of size $O\left(\frac{1}{\varepsilon^2}\log\frac{1}{\varepsilon}\right)$ can be computed deterministically in $O(n)$ time.

We fix $\varepsilon = \frac{\delta}{6}$, and compute for each P_i an ε-approximation subset $A_i \subset P_i$ as above. We then compute a bichromatic deepest line ℓ_A for A_0 and A_1 in $O(1)$ time and return ℓ_A. We now argue that ℓ_A is an almost-deepest line. Observe that for any line ℓ we have (where h ranges over all closed halfspaces containing ℓ)

$$\begin{aligned}\textsc{Depth}(\ell; P_0, P_1) &= \min_h \min_{i=0,1}\{|P_i \cap h|/|P_i|\} \ge \min_h \min_{i=0,1}\{|A_i \cap h|/|A_i|\} - \varepsilon \\ &= \textsc{Depth}(\ell; A_0, A_1) - \varepsilon,\end{aligned}$$

and similarly $\textsc{Depth}(\ell; P_0, P_1) \le \textsc{Depth}(\ell; A_0, A_1) + \varepsilon$.

Let ℓ_A be a bichromatically deepest line for A_0, A_1, and let ℓ_{opt} be a bichromatically deepest line for P_0, P_1. Since $\textsc{Depth}(\ell_{opt}; P_0, P_1) \ge \frac{1}{3}$, we have

$$\begin{aligned}\textsc{Depth}(\ell_A; P_0, P_1) &\ge \textsc{Depth}(\ell_A; A_0, A_1) - \varepsilon \ge \textsc{Depth}(\ell_{opt}; A_0, A_1) - \varepsilon \\ &\ge \textsc{Depth}(\ell_{opt}; P_0, P_1) - \frac{\delta}{3} \ge (1-\delta)\textsc{Depth}_{P_0,P_1}(\ell_{opt}).\end{aligned}$$

We thus conclude the following.

Theorem 5. *For a fixed parameter $\delta > 0$, and two finite point sets $P_0, P_1 \subset \mathbb{R}^3$ with a total of n points, we can compute in $O(n)$ time a line ℓ whose bichromatic depth is at least $1 - \delta$ times the maximum bichromatic depth.*

References

1. P. K. Agarwal. Ray shooting and other applications of spanning trees with low stabbing number. *SIAM J. Comput.*, 21:540–570, 1992.
2. P. K. Agarwal and J. Matoušek. Dynamic half-space range reporting and its applications. *Algorithmica*, 13:325–345, 1995.
3. P. K. Agarwal and M. Sharir. Arrangements and their applications. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 49–119. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

4. P. K. Agarwal, M. Sharir, and E. Welzl. Algorithms for center and Tverberg points. In *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, pages 61–67, 2004. Also to appear in *ACM Trans. Algorithms*.
5. M. Bern and D. Eppstein. Multivariate regression depth. *Discrete Comput. Geom.*, 28(1):1–17, 2002.
6. T. M. Chan. An optimal randomized algorithm for maximum Tukey depth. In *Proc. 15th Annu. ACM-SIAM Sympos. Discrete Algorithms*, pages 430–436, 2004.
7. B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.
8. B. Chazelle. *The Discrepancy Method: Randomness and Complexity.* Cambridge University Press, New York, 2001.
9. B. Chazelle, H. Edelsbrunner, L. Guibas, and M. Sharir. Algorithms for bichromatic line segment problems and polyhedral terrains. *Algorithmica*, 11:116–132, 1994.
10. K. L. Clarkson, D. Eppstein, G. L. Miller, C. Sturtivant, and S.-H. Teng. Approximating center points with iterated Radon points. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 91–98, 1993.
11. T. K. Dey. Improved bounds on planar k-sets and related problems. *Discrete Comput. Geom.*, 19:373–382, 1998.
12. V. Dol'nikov. A generalization of the sandwich theorem. *Mathematical Notes*, 52:771–779, 1992.
13. D. Halperin and M. Sharir. New bounds for lower envelopes in three dimensions, with applications to visibility in terrains. *Discrete Comput. Geom.*, 12:313–326, 1994.
14. S. Jadhav and A. Mukhopadhyay. Computing a centerpoint of a finite planar set of points in linear time. *Discrete Comput. Geom.*, 12:291–312, 1994.
15. J. Matoušek. *Lectures on Discrete Geometry.* Springer Verlag, Berlin, 2002.
16. N. Naor and M. Sharir. Computing a point in the center of a point set in three dimensions. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 10–13, 1990.
17. M. Pellegrini. On lines missing polyhedral sets in 3-space. *Discrete Comput. Geom.*, 12:203–221, 1994.
18. M. Pellegrini. Ray shooting and lines in space. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 37, pages 839–856. CRC Press LLC, Boca Raton, FL, 2nd edition, 2004.
19. P. Rousseeuw and M. Hubert. Depth in an arrangement of hyperplanes. *Discrete Comput. Geom.*, 22:167–176, 1999.
20. P. Rousseeuw and M. Hubert. Regression depth. *J. Amer. Stat. Assoc.*, 94:388–402, 1999.
21. M. Sharir, S. Smorodinsky, and G. Tardos. An improved bound for k-sets in three dimensions. *Discrete Comput. Geom.*, 26:195–204, 2001.
22. M. Sharir and E. Welzl. A combinatorial bound for linear programming and related problems. In *Proc. 9th Annu. Sympos. Theoretical Aspects of Computer Science*, pages 569–579, 1992.
23. G. Toth. Point sets with many k-sets. *Discrete Comput. Geom.*, 26(2):187–194, 2001.
24. R. T. Živaljević and S. T. Vrećica. An extension of the ham sandwich theorem. *Bull. London Math. Soc.*, 22:183–186, 1990.

On Obtaining Pseudorandomness from Error-Correcting Codes[⋆]

Shankar Kalyanaraman and Christopher Umans

Dept of Computer Science, California Institute of Technology, Pasadena, CA 91125
{shankar, umans}@cs.caltech.edu

Abstract. A number of recent results have constructed randomness extractors and pseudorandom generators (PRGs) directly from certain error-correcting codes. The underlying construction in these results amounts to picking a random index into the codeword and outputting m consecutive symbols (the codeword is obtained from the weak random source in the case of extractors, and from a hard function in the case of PRGs).

We study this construction applied to general cyclic error-correcting codes, with the goal of understanding what pseudorandom objects it can produce. We show that *every* cyclic code with sufficient distance yields extractors that fool all linear tests. Further, we show that *every* polynomial code with sufficient distance yields extractors that fool all low-degree prediction tests. These are the first results that apply to univariate (rather than multivariate) polynomial codes, hinting that Reed-Solomon codes may yield good randomness extractors.

Our proof technique gives rise to a systematic way of producing *unconditional* PRGs against restricted classes of tests. In particular, we obtain PRGs fooling all linear tests (which amounts to a construction of ϵ-biased spaces), and we obtain PRGs fooling all low-degree prediction tests.

1 Introduction

Two of the central objects in the area of derandomisation are *extractors* and *pseudorandom generators*. Extractors use a small number of truly random bits to transform "weak" random source into a nearly uniform one. Thus extractors allow the simulation of randomised procedures using only weak randomness (which, for example, may be available from a physical source). In addition to this original motivation, extractors have been used in numerous other settings including complexity theory [Sip88, NZ96, GZ97], algorithms [WZ93], hardness of approximation [Zuc96, Uma99, MU02], distributed protocols [Zuc97, RZ01], and coding theory [TSZ01]. For further discussion see Shaltiel's survey [Sha02]. Quite good constructions of extractors are known now (e.g., [RSW00], [SU05], [LRVW03]), but it remains an open problem to construct optimal extractors.

Pseudorandom generators (PRGs) use a small number of truly random bits to transform a hard function into a small set of strings (a *discrepancy set*) which cannot be distinguished from the uniform distribution by an efficient computational procedure. Thus PRGs prove "hardness vs. randomness" tradeoffs, which show that randomised procedures may be simulated deterministically, under a suitable hardness assumption.

[⋆] This research was supported by NSF grant CCF-0346991 and by BSF grant 2004329.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 105–116, 2006.

A sequence of works has ultimately produced "optimal" PRG constructions [Uma03] that fool general randomised procedures. There is a substantial literature on PRGs that fool more restricted classes of tests, and in some instances *unconditional* constructions (not requiring access to a hard function) are available. For example ϵ-biased spaces [NN93, AGHP92] are PRGs that fool linear tests; other constructions fool affine tests [ABCR97], combinatorial rectangles (see the survey [Sri00]), and general space-bounded computation [Nis92, Nis94, NZ96, INW94, SZ99]. Recently, Bogdanov has constructed PRGs that fool low degree polynomial tests [Bog05].

There is a strong connection between these objects (extractors and PRGs) and *error-correcting codes*. For example, Trevisan's extractor construction [Tre01] uses at its core any good list-decodable code. Subsequent works [TSZS01, SU05] have constructed extractors directly from Reed-Müller codes (and in return, extractors have been used to construct good error-correcting codes in [TSZ01]). PRGs constructed in [STV01, Uma03] have at their core Reed-Müller codes, and it is well-known that ϵ-biased spaces are equivalent to codes with good distance.

In this paper we study a simple construction suited to any cyclic code. Specifically, given any q-ary cyclic error-correcting code $\mathcal{C} : \mathbb{F}_q^{\bar{k}} \to \mathbb{F}_q^{\bar{n}}$, and an additional parameter m, we define the function $f_{\mathcal{C},m} : \mathbb{F}_q^{\bar{k}} \times [\bar{n}] \to \mathbb{F}_q^m$ as follows:

$$f_{\mathcal{C},m}(x, y) = (\mathcal{C}(x)[y + 1], \mathcal{C}(x)[y + 2], \mathcal{C}(x)[y + 3], \ldots, \mathcal{C}(x)[y + m]), \qquad (1)$$

where the symbols of the code are indexed in the cyclic ordering. Our goal is to understand what derandomisation objects are produced by this construction. This construction already has a good "track record" — for certain specific kinds of codes the results of [SU05, Uma03] show that

- $f_{\mathcal{C},m}$ is a (k, ϵ)-extractor with $m = k^{1-\delta}$ when $\mathcal{C}$ is a Reed-Müller code with suitable parameters, and
- $f_{\mathcal{C},m}$ is an ϵ-PRG with $m = k^{\delta}$ when $\mathcal{C}$ is an "augmented" version of a Reed-Müller code with suitable parameters, and when we fix x to be the truth table of a function that cannot be computed by size k circuits.

We are interested in the following questions: Is $f_{\mathcal{C},m}$ a good extractor for *every* cyclic code $\mathcal{C}$ with sufficiently good distance? If so, what parameters does it achieve? What can be said about $f_{\mathcal{C},m}$ when $\mathcal{C}$ is a Reed-Solomon code? Is it a good extractor? Can it be used to produce PRGs against certain restricted classes of tests? We feel that studying the Reed-Solomon code question in particular may illuminate new ways of arguing about code-based extractor constructions (since the local-decodability of Reed-Müller codes that is so heavily relied on in [TSZS01, SU05] is not present in Reed-Solomon codes).

In general these seem to be difficult questions to resolve. In this paper we obtain some modest positive results. Our results are phrased in terms of "fooling" certain classes of tests. Using this terminology, extractors outputting m bits fool the class of all functions from $\{0, 1\}^m$ to $\{0, 1\}$, while PRGs fool the class of all functions from $\{0, 1\}^m$ to $\{0, 1\}$ with small circuits. The proofs for these constructions often transform these "distinguishing" tests into prediction tests (see Section 2 for formal definitions of distinguishers and predictors). In this paper we are concerned with prediction tests directly:

Definition 1. *A* **degree** d **prediction test** *is a degree-d polynomial $p : \mathbb{F}_q^m \to \mathbb{F}_q$ such that p can be expressed as $p(x_1, \ldots, x_m) = x_i - p'(x_1, \ldots, x_{i-1})$ for some i.*

Theorem 1. *Let $\mathcal{C}$ be an $[\bar{n}, \bar{k}, \delta\bar{n}]$ q-ary cyclic linear code with $1^{\bar{n}} \in \mathcal{C}$. For any k and $\rho > 0$, $f_{\mathcal{C},m}$ is a (k, ρ) q-ary extractor for the family of all linear prediction tests, provided that $\delta > 1 - \rho/2$, and $k > m \log q + \log(2/\rho)$.*

When $\mathcal{C}$ is further restricted to be a Reed-Müller code (importantly, including the univariate case, which are Reed-Solomon codes), we show:

Theorem 2. *Let $\mathcal{C}$ be an $[\bar{n}, \bar{k}, \delta\bar{n}]$ q-ary Reed-Müller code with parameters ℓ, h. For any k and $\rho > 0$, $f_{\mathcal{C},m}$ is a (k, ρ) q-ary extractor for the family of all degree d prediction tests, provided that $\rho > 2dh/q$, and $k > m \log q + \log(2/\rho)$.*

Our proofs follow the so-called "reconstruction proof" methodology (see, e.g., [Tre01], [TSZS01], [SU05]). That is, we argue that if the distribution induced by $f_{\mathcal{C},m}$ has a "next-element" predictor of the appropriate type (linear or low-degree), then there is a fixed procedure that "reconstructs" many strings in the weak random source from short advice. This leads to a contradiction, as a source with high min-entropy cannot have many strings that have short descriptions.

Many extractor and PRG constructions employ this proof methodology. From one viewpoint the crucial step is transforming a next-element predictor that errs some fraction of the time into a next-element predictor that is *errorless* (here it becomes clear why error-correcting codes play an important role). From an errorless predictor the remainder of the argument is usually straightforward. From this perspective the main loss associated with the constructions of [SU05], that prevents them from being optimal constructions, is in the conversion from predictors that err to errorless predictors.

Our proofs of Theorem 1 and 2 are noteworthy in that they perform this transformation with *no loss*, for a *wide variety* of codes. Of course the price currently is that we only know how to use this argument to fool a restricted class of tests. Nevertheless, one motivation for exploring these questions, and this methodology in particular, is the possibility of exposing new "lighter-weight" proof techniques that may be useful in the quest to construct optimal extractors.

One consequence of our proof technique is that there is a systematic way to produce *unconditional* PRGs against restricted classes of tests from the above extractor constructions. For example, from the construction in Theorem 1, we obtain a PRG fooling linear prediction tests:

Theorem 3. *Let $\mathcal{C}$ be a systematic $[\bar{n}, \bar{k}, \delta\bar{n}]$ q-ary cyclic linear code with $1^{\bar{n}} \in \mathcal{C}$. Let x be such that $\mathcal{C}(x)[1 \ldots \bar{k}] = 0^{\bar{k}-1}1$. Then $\mathcal{S} = \{f_{\mathcal{C},\bar{k}-1}(x, y) : 1 \leq y \leq \bar{n}\}$ is a q-ary pseudorandom set that fools all linear prediction tests with success probability ρ, provided that $\rho \geq 1 - \delta$.*

By converting the q-ary pseudorandom sets into binary (using a standard idea involving concatenated codes) we get ϵ-biased spaces of size $O(m\text{polylog}(m, 1/\epsilon)/\epsilon^3)$, which are comparable to those one can obtain using the well-known connection to error-correcting codes. By comparison, [NN93] gives a construction of size m/ϵ^c where $4 < c < 5$ while [AGHP92] provides a construction of size $(m/\epsilon)^2$.

Using the same idea, from the construction in Theorem 2, we obtain an unconditional PRG construction that fools low-degree prediction tests:

Theorem 4. *Let $\mathcal{C}$ be a systematic $[\bar{n}, \bar{k}, \delta\bar{n}]$ q-ary cyclic Reed-Müller code with with parameters h, ℓ. Let x be such that $\mathcal{C}(x)[1 \dots \bar{k}] = 0^{\bar{k}-1}1$. Then $\mathcal{S} = \{f_{\mathcal{C},\bar{k}-1}(x, y) : 1 \le y \le \bar{n}\}$ is a q-ary ρ-pseudorandom set for the class of all degree d prediction tests, provided that $\rho \ge dh/q$.*

This construction may sound like it unconditionally derandomises polynomial identity testing. If "prediction tests" were replaced by "distinguishing tests" that would indeed be the case. Yao's Lemma [Yao82] shows how to convert any distinguishing test into a prediction test, but unfortunately it does not preserve low-degree-ness. However our result does derandomise polynomial identity testing for the restricted class of tests that can be phrased as degree d prediction tests.

Of course derandomising polynomial identity testing is a major open problem with significant consequences (see [KI04]). Several works have succeeded in derandomising polynomial identity testing for restricted classes of polynomials ([DS05], [RS], [LV98]). Our result derandomises polynomial identity testing for degree d prediction tests; in fact it produces a stronger object, a *hitting set with density* $1 - \rho$ (see the discussion following Definition 5). We don't know of any trivial constructions of hitting sets with density $1-\rho$ for even this simple class of polynomials, making it an interesting testbed for new techniques.

Two other works construct hitting sets with density $1 - \rho$ for general classes of polynomials: Bogdanov [Bog05] constructs a hitting set of density $1 - \rho$ against all m-variate polynomials of degree d, of size $m^{O(d \log(d/\rho))}$. Klivans and Spielman [KS01] construct hitting sets of density $1-\rho$ against all m-variate polynomials of degree d with M monomials, of size $O(mMd/\rho)$. Our construction has a much smaller size, md/ρ, against a particular subclass of m-variate polynomials of degree d (degree d prediction tests). For many settings of the parameters, this is an exponential improvement, albeit for a limited class of polynomials. It is in fact surprising that we obtain hitting sets of this size without an explicit constraint on the size of an arithmetic circuit computing the polynomial.

2 Preliminaries

Two distributions P and Q over a finite set S are said to be ϵ-close if their ℓ_1-distance given by $\sum_{x \in S} |P(x) - Q(x)|$ is at most 2ϵ or equivalently if $\max_{A \subseteq S} |P(A) - Q(A)|$ is at most ϵ. The min-entropy of a random variable X with distribution P on S is defined as $H_\infty(X) = \min_{x \in S} \log(1/P(x))$. We often use U_n as a uniformly distributed random variable.

Definition 2. *A* **distinguisher** with advantage ϵ *for a random variable $X = (X_1, X_2, \dots, X_m)$ defined on $\mathbb{F}_q^m$ is a function $f : \mathbb{F}_q^m \to \mathbb{F}_q$ with the property that*

$$|Pr[f(X) = 0] - Pr[f(U_m) = 0]| \ge \epsilon$$

where U_m is uniformly distributed on $\mathbb{F}_q^m$.

Definition 3. *An* i^{th}**-element predictor** *with success probability* ρ *for a random variable* $X = (X_1, X_2, \ldots, X_m)$ *defined on* $\mathbb{F}_q^m$ *is a function* $f : \mathbb{F}_q^{i-1} \rightarrow \mathbb{F}_q$ *such that:* $Pr[f(X_1, \ldots, X_{i-1}) = X_i] \geq \rho$*. If* $\rho = 1$ *we say that* f *is* errorless.

We will be concerned with linear and low-degree distinguishers and predictors. Note that a linear function f satisfies the identities (i) $f(\sum_{j=1}^{k} x_j) = \sum_{j=1}^{k} f(x_j) - (k-1)f(0)$ and (ii) $f(\alpha x) = \alpha f(x) - (\alpha - 1)f(0)$ for any scalar α. A *homogeneous* linear function f has $f(0) = 0$.

Definition 4. *A* (k, ρ) q**-ary extractor for a family of predictors** $\mathcal{P}$ *is a function* $E : \{0,1\}^n \times \{0,1\}^t \rightarrow \mathbb{F}_q^m$ *such that for every random variable* X *with* $H_\infty(X) \geq k$*, there is no* i^{th}*-element predictor* $f \in \mathcal{P}$ *for* $E(X, U_t)$ *with success probability* ρ *for any* $i = 1, \ldots m$*.*

In our notation, the usual q-ary extractors (as defined in, e.g., [SU05]) are simply q-ary extractors for the family of all predictors. Rather than referring to PRGs directly we prefer to describe the set of strings they produce.

Definition 5. *A* q*-ary* ρ**-pseudorandom set for a family of predictors** $\mathcal{P}$ *is a multiset* S *such that there is no* i*-th element predictor* $f \in \mathcal{P}$ *with success probability* ρ *for the random variable induced by picking an element uniformly at random from* S*.*

In Bogdanov's terminology [Bog05], a ρ-pseudorandom set for a family of predictors $\mathcal{P}$ is called a *hitting set with density* $1 - \rho$ for the class of degree d prediction tests[1]. In fact, it is a simple observation that a ρ-pseudorandom set S for the family of degree d predictors has the property that for every degree d prediction test g, the distribution $g(Z)$ is ρ-close to the distribution $g(X)$ in the max-norm (where Z is a random variable uniformly distributed on S, and X is a uniform random variable). One can also ask that $g(Z)$ and $g(X)$ be ρ-close in the ℓ_1 norm. This gives rise to genuinely a stronger object, termed a *pseudorandom generator of bias* ρ in [Bog05].

Definition 6. *An* $[\bar{n}, \bar{k}, \bar{d}]$ q**-ary linear code** *is a subspace* $\mathcal{C} \subseteq \mathbb{F}_q^{\bar{n}}$ *for which the Hamming distance between every pair* $x, y \in \mathcal{C}$ *is at least* $\bar{d}$*.*

Given a string x, we will often use $C(x)$ to mean the x-th codeword in C (and all of the codes we consider come equipped with efficient ways to compute this encoding function). A code is *systematic* if the message appears as a prefix of every codeword.

Definition 7. *A code* $\mathcal{C}$ *is* **cyclic** *if it satisfies the following condition:*

$$(x_1, x_2, \ldots, x_{\bar{n}-1}, x_{\bar{n}}) \in \mathcal{C} \Rightarrow (x_{\bar{n}}, x_1, x_2, \ldots, x_{\bar{n}-1}) \in \mathcal{C}.$$

We always treat the indices into a cyclic code modulo $\bar{n}$*.*

A specific family of q-ary codes we will use are the Reed-Müller codes. The codewords of a *Reed-Müller code with parameters* ℓ, h are the evaluations of ℓ-variate polynomials of total degree at most h, at the points $\mathbb{F}_q^\ell \backslash \{0\}$. The special case of $\ell = 1$ gives the *Reed-Solomon codes*. All of these codes are cyclic (for an appropriate ordering of $\mathbb{F}_q^\ell \setminus \{0\}$) and linear.

[1] A *hitting set of density* α *for a family of functions* $\mathcal{F}$ is a multiset $H \subseteq \mathbb{F}_q^m$ such that for every non-zero function $p \in \mathcal{F}$, $\Pr_{x \in H}[p(x) \neq 0] > \alpha$.

3 Overview of the Results

In this section we describe the high-level ideas behind our results, before giving the technical details and full proofs in the next section.

3.1 Extractors Fooling Linear Tests

Let $\mathcal{C}$ be any cyclic code, and consider the function $f_{\mathcal{C},m}$ from (1). We show that for fixed x, if the distribution $f_{\mathcal{C},m}(x, y)$ with y chosen uniformly at random has a linear predictor p, then x has a short description. In this case p is a linear function for which:

$$p(\mathcal{C}(x)[y + 1], \mathcal{C}(x)[y + 2], \ldots, \mathcal{C}(x)[y + m - 1]) = \mathcal{C}(x)[y + m] \tag{2}$$

with noticeable probability over the choice of y.

Our key observation is that if $\mathcal{C}$ has sufficiently good distance, then f must be *errorless*. To prove this we first select a subset S of those y for which (2) holds. If $\mathcal{C}$ has sufficiently good distance, then a given position r may be expressed as a linear combination ℓ of the values of $\mathcal{C}(x)$ at the positions S:

$$\mathcal{C}(x)[r] = \ell(\mathcal{C}(x)[y])_{y\in S}$$

Since $\mathcal{C}$ is *cyclic*, this same equation holds for *every* cyclic shift; i.e., for all i: $\mathcal{C}(x)[r + i] = \ell(\mathcal{C}(x)[y + i])_{y\in S}$. These equations together with (2), which holds for all $y \in S$, imply that (2) holds for r. Since r was arbitrary, we conclude that p is indeed errorless.

From here, it is easy to see that x may be described by $\mathcal{C}(x)[1 \ldots m - 1]$, since we can use p to obtain $\mathcal{C}(x)[m]$, and again to obtain $\mathcal{C}(x)[m + 1]$, and so on, until we have $\mathcal{C}(x)$ in its entirety. Finally decoding $\mathcal{C}(x)$ recovers x.

Note that "extractors that fool linear tests" are not meaningful in the usual setting of simulating randomised procedures using a weak random source. This is because if one is only trying to fool linear tests, one could use ϵ-biased spaces to do away with the randomness altogether. However, we believe that this setting is a good testbed for refining the "reconstruction proof" technique, and that it may be valuable to adapt it in the way we do here, to obtain an errorless predictor without relying on local-decodability of the underlying code. Additionally, our goal is to understand the construction in (1) in the most general setting possible, and the fact that an extractor object (albeit against a restricted class of tests) is produced from *any* cyclic code is a step toward that goal.

3.2 Extractors Fooling Low-Degree Tests

Now suppose further that $\mathcal{C}$ is a *polynomial* cyclic code; i.e., a Reed-Müller code, and we have the same setup except that the predictor p is now only *low degree*. That is, there is a function p of degree d for which:

$$p(\mathcal{C}(x)[y + 1], \mathcal{C}(x)[y + 2], \ldots, \mathcal{C}(x)[y + m - 1]) = \mathcal{C}(x)[y + m] \tag{3}$$

with noticeable probability over the choice of y. The argument used for linear p breaks down, but a different argument works, relying on the fact that $\mathcal{C}(x)$ is now itself a low-degree polynomial. This means that there is a mapping between the index y and values for variables $y_1, y_2, \ldots, y_n$ for which $r_x(y_1, y_2, \ldots, y_n) \equiv \mathcal{C}(x)[y]$, where r_x is a low-degree polynomial depending on x. The fact that $\mathcal{C}$ is cyclic means that for all i there is a low-degree polynomial $r_{x,i}$ for which $r_{x,i}(y_1, y_2, \ldots, y_n) \equiv \mathcal{C}(x)[y + i]$.

Now, we observe that the left-hand side of (3) is a low-degree polynomial in $y_1, y_2, \ldots, y_n$, as is the right hand side. However, they agree with noticeable probability, so for an appropriate choice of parameters, they must be *equal*. This implies that p is an *errorless* predictor, since equation (3) holds for all y.

3.3 Unconditional PRGs Fooling Linear and Low-Degree Tests

If for a given code $\mathcal{C}$, one can identify a fixed "good" x for which $f_{\mathcal{C},m}(x, \cdot)$ fools all *efficient* predictors, then $f_{\mathcal{C},m}(x, \cdot)$ generates a discrepancy set against all small circuits. It is standard that such an x yields a function that is not computable by small circuits, and thus in the absence of strong circuit lower bounds we can obtain (at best) a *conditional* construction. When the class of predictors is restricted in a different way, we can pursue the same strategy to produce a *pseudorandom set* against all predictors in this class.

One of the surprising side-effects of having transformations from a predictor to an errorless predictor like the ones we have is that it is easy to produce a "good" x, *unconditionally*. This is because we need only to find a codeword that cannot have an errorless predictor. In fact, any codeword beginning with 0^m1, will suffice. If such a codeword has an errorless predictor p, then that predictor must output 0 since

$$p(\mathcal{C}(x)[y + 1], \mathcal{C}(x)[y + 2], \ldots, \mathcal{C}(x)[y + m - 1]) = \mathcal{C}(x)[y + m]$$

implies $p(0, 0, 0 \ldots, 0) = 0$ (when $y = 0$) and $p(0, 0, 0 \ldots, 0) = 1$ (when $y = 1$), a contradiction. This gives a simple construction of pseudorandom sets fooling all linear tests from any cyclic code with good distance. We are also able to conclude that substrings of low-degree polynomials comprise a pseudorandom set that fools low-degree prediction tests, giving a derandomisation of polynomial identity testing for this restricted class of tests.

4 Proofs of Main Results

In this section, we shall provide a formal proof of Theorem 1 and sketch proofs of the remaining main theorems.

4.1 Extractors Fooling Linear Tests

We present a construction for q-ary extractors that fool all linear prediction tests. We begin with a crucial property of linear codes (the proof appears in the full version):

Lemma 1. *Let $\mathcal{C}$ be an $[\bar{n}, \bar{k}, \bar{d}]$ q-ary linear code. Let $S = \{t_1, \ldots, t_m\} \subseteq \{1, 2, \ldots, n\}$ be a set of size at least $\bar{n} - \bar{d} + 1$, and pick $r \in \{1, 2, \ldots n\}$. Then there exists a homogeneous linear function $f : F_q^m \to \mathbb{F}_q$ such that for all x, $\mathcal{C}(x)[r] = f(\mathcal{C}(x)[t_1], \ldots, \mathcal{C}(x)[t_m])$.*

We prove that a "reasonably correct" linear predictor operating on a codeword in a suitable code must in fact be exactly correct.

Lemma 2. *Let $\mathcal{C}$ be an $[\bar{n}, \bar{k}, \delta\bar{n}]$ q-ary cyclic linear code with $1^{\bar{n}} \in \mathcal{C}$, and fix x. Suppose p is a linear i^{th}-element predictor with success probability $\rho > (1-\delta)$ for the random variable $f_{\mathcal{C},m}(x,y)$ induced by picking y uniformly from $\{1, 2, \ldots, \bar{n}\}$. Then, p is an errorless linear predictor.*

Proof. Define S to be the set of positions on which p is correct; i.e.,

$$S = \{s : p(\mathcal{C}(x)[s+1], \mathcal{C}(x)[s+2], \ldots, \mathcal{C}(x)[s+i-1]) = \mathcal{C}(x)[s+i]\}$$

We know that $|S| \geq (1-\delta)\bar{n} + 1$. Now pick an arbitrary $r \in \{1, 2, \ldots, \bar{n}\}$, and let $f = \sum_{s \in S} \alpha_s z_s$ be the linear function guaranteed by Lemma 1. We have: $p(\mathcal{C}(x)[r+1], \ldots, \mathcal{C}(x)[r+i-1]) = p\left(\sum_{s\in S} \alpha_s \mathcal{C}(x)[s+1], \ldots, \sum_{s\in S} \alpha_s \mathcal{C}(x)[s+i-1]\right)$

$$= \sum_{s\in S} \alpha_s p(\mathcal{C}(x)[s+1], \ldots, \mathcal{C}(x)[s+i-1]) + \left(1 - \sum_{s\in S} \alpha_s\right) p(0, \ldots 0)$$

$$= \sum_{s\in S} \alpha_s p(\mathcal{C}(x)[s+1], \ldots, \mathcal{C}(x)[s+i-1]) = \sum_{s\in S} \alpha_s \mathcal{C}(x)[s+i] = \mathcal{C}(x)[r+i]$$

where the second line follows from the fact that p is linear (using two properties of linear functions noted in Section 2), and the third line follows because $1^{\bar{n}} \in \mathcal{C}$ implies $(1 - \sum_{s\in S} \alpha_s) = 0$, and from the definition of S. □

We now prove our first main theorem, showing that $f_{\mathcal{C},m}$ for cyclic codes $\mathcal{C}$ is an extractor fooling q-ary linear tests.

Proof (of Theorem 1). Suppose $f_{\mathcal{C},m}$ is not an extractor with the parameters as claimed. Then there is some random variable X having distribution D, with min-entropy at least k, and for some i, a linear i^{th}-element predictor p satisfying

$$\Pr_{x \leftarrow D, y}[p(f_{\mathcal{C},m}(x,y)_{1,\ldots,i-1}) = f_{\mathcal{C},m}(x,y)_i] \geq \rho.$$

By an averaging argument

$$\Pr_{x \leftarrow D}[\Pr_y[p(f_{\mathcal{C},m}(x,y)_{1,\ldots,i-1}) = f_{\mathcal{C},m}(x,y)_i] \geq \rho/2] \geq \rho/2. \quad (4)$$

Now, for every x for which $\Pr_y[p(f_{\mathcal{C},m}(x,y)_{1,\ldots,i-1}) = f_{\mathcal{C},m}(x,y)_i] \geq \rho/2$, Lemma 2 implies that $\Pr_y[p(f_{\mathcal{C},m}(x,y)_{1,\ldots,i-1}) = f_{\mathcal{C},m}(x,y)_i] = 1$, since $\rho/2 > 1-\delta$. Every such x can be described with $(i-1)$ elements of $\mathbb{F}_q$, by simply writing down $\mathcal{C}(x)[1\ldots, i-1]$. From this, $p(\mathcal{C}(x)[1\ldots, i-1]) = \mathcal{C}(x)[i]$, and then $p(\mathcal{C}(x)[2\ldots, i] = \mathcal{C}(x)[i], p(\mathcal{C}(x)[3\ldots, i+1] = \mathcal{C}(x)[i+2]$, and so on until we obtain all of the symbols of $\mathcal{C}(x)$, which in turn determine x.

We can define a function $R : \mathbb{F}_q^{i-1} \rightarrow \mathbb{F}_q^{\bar{k}}$ that runs this procedure. Using equation (4) above, we get: $\Pr_{x\leftarrow D}[\exists a \in \mathbb{F}_q^{i-1}$ for which $R(a) = x] \geq \rho/2$. A given x is sampled with probability at most 2^{-k}, and so applying the union bound, the probability above is bounded above by $q^{i-1}2^{-k}$. Using the fact that $i \leq m$, we get a contradiction if $2^{m\log q - k} < \rho/2$, or equivalently $k > m\log q + \log(2/\rho)$. Our choice of k thus implies that $f_{\mathcal{C},m}$ must be the claimed extractor. □

To get a sense of the achievable extractor parameters here, we plug in a Reed-Müller code:

Corollary 1. *Fix n, k, and $\rho > 1/k^{O(1)}$. Let $\mathcal{C}$ be a Reed-Müller code with parameters $h = k$, $q = 2k/\rho$, and $\ell = \log n / \log k$. Then $f_{\mathcal{C},m}$ is a (k, ρ) q-ary extractor for the family of all linear prediction tests, with seed length $O(\log n)$ and output length $m \geq k/O(\log k)$.*

4.2 Extractors Fooling Low-Degree Tests

We develop our result further to describe constructions of extractors that fool low-degree prediction tests. While the extractor constructions presented in the previous section can be derived in general from any cyclic, linear code the following constructions are obtained from Reed-Müller codes (including the special case of Reed-Solomon codes). Similar to the previous subsection, we present a lemma that says that a "reasonably good" low-degree predictor is an errorless low-degree predictor.

Lemma 3. *Let $\mathcal{C}$ be an $[\bar{n}, \bar{k}, \bar{d}]$ q-ary Reed-Müller code with parameters ℓ, h, and fix x. Suppose p is a degree d i-th element predictor with success probability $\rho > dh/q$ for the random variable $f_{\mathcal{C},m}(x, y)$ induced by picking y uniformly from $\{1, 2, \ldots, \bar{n}\}$. Then p is an errorless predictor.*

We omit the proof for lack of space, but note that it is described informally in Section 3.2. This gives us Theorem 2, showing that $f_{\mathcal{C},m}$ for Reed-Müller codes $\mathcal{C}$ is an extractor fooling low-degree tests. The proof uses Lemma 3 in similar fashion to how the proof of Theorem 1 uses Lemma 2.

The following corollary plugs in Reed-Solomon codes, which correspond to $\ell = 1$ in Theorem 2.

Corollary 2. *Fix n, k, d and $\rho > 1/k^{O(1)}$. Let $\mathcal{C}$ be a q-ary Reed-Solomon code with parameters $q = 2dn/\rho$ and $h = n$. Then $f_{\mathcal{C},m}$ is a (k, ρ) q-ary extractor for the family of all degree d prediction tests, with seed length $O(\log n)$ and output length $m \geq k/O(\log dn)$.*

5 Pseudorandom Sets for Linear and Low-Degree Tests

In this section we obtain unconditional PRGs by using a special feature of our proof methodology.

5.1 Pseudorandom Sets for Linear Tests

Using Lemma 2 again, we can prove Theorem 3 giving a construction of a pseudorandom set for linear prediction tests using any systematic cyclic linear code as described in Section 3.3. Specifically, using a systematic Reed-Solomon code gives us Corollary 3 and furthermore, using Proposition 1 gives us pseudorandom sets fooling linear distinguishing tests in Corollary 4, all of which are stated below.

Corollary 3. *Fix m, ρ. Let $\mathcal{C}$ be a systematic Reed-Solomon code with parameters h, q satisfying $q = h/\rho$. The set $\mathcal{S}$ described in Theorem 3 is a q-ary ρ-pseudorandom set in $\mathbb{F}_q^m$ of size h/ρ for the class of all linear prediction tests.*

Proposition 1. *Let $f : \mathbb{F}_q^m \rightarrow \mathbb{F}_q$ be a q-ary linear distinguisher for a distribution D with advantage ϵ. Then, there exists an i and a q-ary linear next-element predictor for D f' such that for a random variable x defined over $\mathbb{F}_q^m$, $\Pr_{x\leftarrow D}[f'(x_1,\ldots,x_{i-1}) = x_i] \geq \frac{1}{q}+\frac{\epsilon}{q-1}$ and for the case $\frac{1}{q} \leq \epsilon \leq 1-\frac{1}{q}$, $\Pr_{x\leftarrow D}[f'(x_1,\ldots,x_{i-1}) = x_i] \geq \frac{1}{q}+\epsilon$.*

We reserve the proof of the proposition for the full version of the paper.

Corollary 4. *Let $\mathcal{C}$ and S be as defined above in Theorem 3. For every $v \in \mathbb{F}_q^m$, $|\Pr_{s\in S}[s \cdot v = 0] - \Pr_x[x \cdot v = 0]| \leq \left(\rho - \frac{1}{q}\right)(q-1)$.*

Pseudorandom sets for binary linear distinguishing tests are called ϵ-biased sample spaces. Using our constructions from above and combining them with good binary codes we can construct good ϵ-biased sample spaces.

Definition 8. *A multiset $\mathcal{T} \subseteq \{0,1\}^m$ is an* ϵ**-biased sample space** *if for every $\boldsymbol{v} \in \{0,1\}^m$, $|\Pr_{x\in T}[x \cdot v = 0] - \Pr_{x\in T}[x \cdot v = 1]| \leq \epsilon$.*

Theorem 5. *Let $\mathcal{C}_1$ be an $[\bar{n}_1, \bar{k}_1, \delta_1\bar{n}_1]$ q-ary cyclic code, and $\mathcal{C}_2$ be an $[\bar{n}_2, \bar{k}_2 = \log q, \delta_2\bar{n}_2]$ binary systematic code, and set $m = \bar{k}_1 - 1$. Define $\mathcal{S} = \{f_{\mathcal{C}_1,m}(x,y) : 1 \leq y \leq \bar{n_1}\}$ and define*

$$\mathcal{T} = \{(\mathcal{C}_2(s_1)[z], \mathcal{C}_2(s_2)[z], \ldots, \mathcal{C}_2(s_m)[z]) : (s_1, s_2, \ldots, s_m) \in S, z \in \{1, 2, \ldots, \bar{n}_2\}\}$$

The set $\mathcal{T}$ is a 4ϵ-biased sample space, provided $\delta_1 > 1 - \epsilon$, and $\delta_2 > 1/2 - \epsilon$.

We defer the proof to the full version of the paper. For fixed m, ϵ, if we choose $\mathcal{C}_1$ to be a $[q, m+1, q-m]$ RS-code where $q > m/\epsilon$ and $\mathcal{C}_2$ to be a $[q, \log q, q/2]$ binary Hadamard code we can apply Theorem 5 to get a 4ϵ-biased space of size $O(m^2/\epsilon^2)$. Moreover, with $\mathcal{C}_2$ as an $[\bar{n} = O(\log q^2/\epsilon^2), \log q, (1/2-\epsilon)\bar{n}]$ binary code, we improve the size to $O(m\text{polylog}(m, 1/\epsilon)/\epsilon^3)$.

5.2 Pseudorandom Sets for Low-Degree Tests

We extend the previous discussion to pseudorandom sets for low-degree tests derived from Reed-Müller codes. In similar fashion to proving Theorem 3, we can prove Theorem 4 using Lemma 3 and arguing that a good prediction test would imply an errorless prediction test which, given the construction is impossible, hence contradicting our assumption. As before, by using a specific Reed-Solomon code we obtain Corollary 5.

Corollary 5. *Fix m, ρ. Let $\mathcal{C}$ be a systematic Reed-Solomon code with parameters h, q satisfying $q = dh/\rho$. The set $\mathcal{S}$ described in Theorem 4 is a q-ary ρ-pseudorandom set in $\mathbb{F}_q^m$ of size hd/ρ for the class of all degree d prediction tests.*

Equivalently, we have an explicit construction of a hitting set with density $1-\rho$ against degree d prediction tests, with size md/ρ. As discussed in the introduction this is somewhat surprising. Even for this simple class of polynomials, there does not seem to be a trivial construction of a hitting set with density $1-\rho$, making Theorem 4 another example where the generic object $f_{C,m}$ yields a non-trivial pseudorandom construction.

6 Concluding Remarks

There are many questions raised by these results. For example, is it possible to enlarge the class of tests fooled by the extractors and pseudorandom sets constructed from arbitrary cyclic linear codes? Similarly, is it possible to fool more general prediction tests using arbitrary polynomial codes? The results of [SU05] show that it is in the particular case of Reed-Müller codes (with certain parameters), but it is possible that something more general is true depending, e.g., only on the distance of the code.

We feel that one of the nicest questions of this type is the question of whether $f_{\mathcal{C},m}$ is a extractor (fooling all prediction tests), when $\mathcal{C}$ is a Reed-Solomon code.

Regarding pseudorandom sets for low-degree polynomials, we wonder if there is a nontrivial conversion of distinguishers to predictors (probably relying on the distinguisher being presented as a small arithmetic circuit) that preserves low-degree-ness. This would potentially lead to a non-trivial derandomisation of polynomial identity testing, because it would imply that the pseudorandom sets of Theorem 4 would in fact fool low-degree distinguishing tests with small circuits.

Acknowledgements. We thank Eli Ben-Sasson for helpful discussions and Andrej Bogdanov for sharing a draft of [Bog05] with us. We also thank the anonymous referees for their insightful comments.

References

[ABCR97] A. E. Andreev, J. L. Baskakov, A. E. F. Clementi, and J. D. P. Rolim. Small random sets for affine spaces and better explicit lower bounds for branching programs. Technical Report TR04-053, ECCC, 1997.

[AGHP92] N. Alon, O. Goldreich, J. Hastad, and R. Peralta. Simple constructions of almost k-wise independent random variables. *Random Struct. Algorithms*, (3):289–304, 1992.

[Bog05] A. Bogdanov. Pseudorandom generators for low degree polynomials. In *Proceedings of STOC*, pages 21–30, 2005.

[DS05] Z. Dvir and A. Shpilka. Locally decodable codes with 2 queries and polynomial identity testing for depth 3 circuits. In *Proceedings of STOC*, pages 592–601, 2005.

[GZ97] O. Goldreich and D. Zuckerman. Another proof that BPP subseteq PH (and more). Technical Report TR97-045, ECCC, 1997.

[INW94] R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proceedings of STOC*, pages 356–364, 1994.

[KI04] V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity tests means proving circuit lower bounds. volume 13, pages 1–46, 2004.

[KS01] A. Klivans and D. Spielman. Randomness efficient identity testing of multivariate polynomials. In *Proceedings of STOC*, pages 216–223, 2001.

[LRVW03] C.-J. Lu, O. Reingold, S. Vadhan, and A. Wigderson. Extractors: optimal up to constant factors. In *Proceedings of STOC*, pages 602–611, 2003.

[LV98] D. Lewin and S. Vadhan. Checking polynomial identities over any field: Towards a derandomization? In *Proceedings of STOC*, pages 438–447, 1998.

[MU02] E. Mossel and C. Umans. On the complexity of approximating the VC dimension. *J. Comput. Syst. Sci.*, 65(4):660–671, 2002.

[Nis92] N. Nisan. Pseudorandom generators for space-bounded computation. *Combinatorica*, 12:249–461, 1992.

[Nis94] N. Nisan. $\mathrm{RL} \subseteq \mathrm{SC}$. *Computational Complexity*, 4(1):1–11, 1994.

[NN93] J. Naor and M. Naor. Small-bias probability spaces: Efficient constructions and applications. *SICOMP*, 22(4):838–856, August 1993.

[NZ96] N. Nisan and D. Zuckerman. Randomness is linear in space. *J. Comput. Syst. Sci.*, 52(1):43–52, February 1996.

[RS] R. Raz and A. Shpilka. Deterministic polynomial identity testing in non-commutative models. In *CCC*.

[RSW00] O. Reingold, R. Shaltiel, and A. Wigderson. Extracting randomness via repeated condensing. In IEEE, editor, *FOCS*, pages 22–31, 2000.

[RZ01] A. Russell and D. Zuckerman. Perfect information leader election in $\log^* n + O(1)$ rounds. *J. Comput. Syst. Sci.*, 63(4):612–626, 2001.

[Sha02] R. Shaltiel. Recent developments in explicit constructions of extractors. *Bulletin of EATCS*, 77:67–95, June 2002. Columns: Computational Complexity.

[Sip88] M. Sipser. Expanders, randomness, or time versus space. *J. Comput. Syst. Sci.*, 36(3):379–383, 1988.

[Sri00] A. Srinivasan. Low-discrepancy sets for high-dimensional rectangles: a survey. *Bulletin of the EATCS*, 70:67–76, 2000.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *J. Comput. Syst. Sci.*, 62(2):236–266, March 2001.

[SU05] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *J. ACM*, 52(2):172–216, 2005.

[SZ99] M. Saks and S. Zhou. BPSPACE(S) $\subseteq$ DSPACE($S^{3/2}$). *J. Comput. Syst. Sci.*, 58(2):376–403, 1999.

[Tre01] L. Trevisan. Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879, July 2001.

[TSZ01] A. Ta-Shma and D. Zuckerman. Extractor codes. In ACM, editor, *Proceedings of STOC*, pages 193–199, 2001.

[TSZS01] A. Ta-Shma, D. Zuckerman, and S. Safra. Extractors from Reed-Muller codes. In *Proceedings of FOCS*, pages 638–647, 2001.

[Uma99] C. Umans. Hardness of approximating Σ_2^p minimization problems. In *Proceedings of FOCS*, pages 465–474, 1999.

[Uma03] C. Umans. Pseudo-random generators for all hardnesses. *J. Comput. Syst. Sci.*, 67(2):419–440, 2003.

[WZ93] A. Wigderson and D. Zuckerman. Expanders that beat the eigenvalue bound: Explicit construction and applications. In *Proceedings of STOC*, pages 245–251, 1993.

[Yao82] A. C. Yao. Theory and applications of trapdoor functions. In *Proceedings of the 23rd Symposium on Foundations of Computer Science (FOCS)*, pages 80–91. IEEE Computer Society Press, 1982.

[Zuc96] D. Zuckerman. On unapproximable versions of NP -complete problems. *SICOMP*, 25(6):1293–1304, December 1996.

[Zuc97] D. Zuckerman. Randomness-optimal oblivious sampling. *Random Struct. Algorithms*, 11:345–367, 1997.

Fast Edge Colorings with Fixed Number of Colors to Minimize Imbalance

Gruia Calinescu[1],* and Michael J. Pelsmajer[2]

[1] Department of Computer Science, Illinois Institute of Technology,
Chicago, IL 60616, USA
calinescu@iit.edu

[2] Department of Applied Mathematics, Illinois Institute of Technology,
Chicago, IL 60616, USA
pelsmajer@iit.edu

Abstract. We study the following optimization problem: the input is a multigraph $G = (V, E)$ and an integer parameter g. A feasible solution consists of a (not necessarily proper) coloring of E with colors $1, 2, \ldots, g$. Denote by $d(v, i)$ the number of edges colored i incident to v. The objective is to minimize $\sum_{v \in V} \max_i d(v, i)$, which roughly corresponds to the "imbalance" of the edge coloring. This problem was proposed by Berry and Modiano (INFOCOM 2004), with the goal of optimizing the deployment of tunable ports in optical networks. Following them we call the optimization problem MTPS - Minimum Tunable Port with Symmetric Assignments.

Among other results, they give a reduction from Edge Coloring showing that MTPS is NP-Hard and then give a 2-approximation algorithm. We give a (3/2)-approximation algorithm. Key to this problem is the following question: given a multigraph $G = (V, E)$ of maximum degree g, what fraction of the vertices can be properly edge-colored in a coloring with g colors, where a vertex is properly edge-colored if the edges incident to it have different colors? Our main lemma states that there is such a coloring with half of the vertices properly edge-colored. For $g \leq 4$, two thirds of vertices can be made properly edge-colored.

Our algorithm is based on g Maximum Matching computations (total running time $O(gm\sqrt{n + m/g})$) and a local optimization procedure, which by itself gives a 2-approximation. An interesting analysis gives an expected $O((gn + m)\log(gn + m))$ running time for the local optimization procedure.

Keywords: Approximation Algorithms, Graph Theory, Edge Coloring, Randomized Algorithms

1 Introduction

Berry and Modiano [2] study the benefits of using tunable electronic ports in WDM/TDM Optical Networks. They provide formulations for the "tunable"

* Research partially supported by NSF grant CCF-0515088.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 117–128, 2006.

optimization problems of reducing the number of tunable electronic ports. These ports are very expensive and optimal placement is very desirable.

They introduce two optimization problem. In this paper we concentrate on the Minimum Tunable Port with Symmetric Assignments (MTPS) problem: The input is a multigraph $G = (V, E)$ and an integer parameter g. A feasible solution consists of a (not necessarily proper) coloring (called a *g-edge coloring*) of E with colors $1, 2, \ldots, g$. Denote by $d(v, i)$ the number of edges colored i incident to v. The objective is to minimize $\sum_{v \in V} \max_i d(v, i)$.

Actually, the MTPS problem as described in [2] has a different description. They give a non-trivial equivalence reduction to the formulation above, which they use for proving NP-Completeness and for approximation algorithms. For $g = 3$, they show the problem is NP-Complete by an easy reduction from Edge Coloring in cubic graphs [7]. Indeed, one can see that a proper 3-edge coloring (that is, a coloring with 3 colors where no two edges incident to a vertex have the same color) of a cubic graph is the only way a 3-edge coloring can have objective function in MTPS equal to $|V|$. The result of [7] can be used (though we do not prove this here) to show that MTPS is APX-Hard: that is no $(1+\epsilon)$-approximation algorithm exists unless P=NP [1].

An edge coloring is called *equitable* [6] if for all vertices v and colors i, j, we have $d(v, i) \leq d(v, j) + 1$. It is clear that an equitable edge coloring, if it exists, minimizes the objective function [3]. Certain classes of graphs, for example simple graphs where no vertex has degree multiple of g, are known to have equitable g-edge colorings [6].

Berry and Modiano [2] give a conceptually simple 2-approximation algorithm, which we describe later. We give a (3/2)-approximation algorithm. Key to the MTPS problem is the following question: given a graph $G = (V, E)$ of maximum degree g, what fraction of the vertices can be *properly edge-colored* in a g-edge coloring, where a vertex is properly edge-colored (or just *proper*) if the edges incident to it have different colors? Our main lemma states that there is a g-edge coloring with half of vertices properly edge-colored. For $g \leq 4$, two thirds of vertices can be made properly edge-colored; this bound is tight. We leave as an open question the problem of finding tight bounds for larger values of g.

Our algorithm for $g > 4$ is based on g Maximum Matching computations (total running time $O(gm\sqrt{n + m/g})$) and a local optimization procedure, which by itself gives a 2-approximation. By carefully implementing this local optimization procedure, we prove it has an expected $O((gn + m)\log(gn + m))$ running time. This implementation is needed to ensure the overall running time is not dominated by local optimization. Local optimization would be a top choice of a practitioner, and depending on the size of the instance, it may be important to have a fast implementation. For $g = 3$ and $g = 4$ we obtain a 4/3-approximation algorithm with running time of $O((n + m)\log n)$ and $O(n^2 + m^2)$, respectively.

A related problem was considered by Feige et. al. [5] In Maximum Edge Coloring, given a multigraph $G = (V, E)$ and a parameter g, one seeks a subgraph with maximum number of edges which can be properly edge-colored with g colors. They show that Maximum Edge Coloring is Max-SNP Hard, even for $g = 2$, and give constant approximation algorithms.

Our paper is organized as follow. The next section presents preliminaries, the case $g = 2$, introduces the local optimization procedure, and gives two simple 2-approximation algorithms. Section 3 gives the approximation ratio of the algorithms, ignoring implementation details and the analysis of the running time of the algorithms, which appear in Section 4.

2 Preliminaries

All our graphs are multigraphs, unless stated otherwise. For $F \subseteq E(G)$, we use $G \setminus F$ to denote the graph $(V(G), E(G) \setminus F)$. For $A \subseteq V(G)$, we use $G \setminus A$ to denote the subgraph of G induced by $V(G) \setminus A$.

The obvious lower bound for the optimum is $L = \sum_{v \in V} \lceil \frac{d(v)}{g} \rceil$, where $d(v)$ is the degree of vertex v. Berry and Modiano [2] use L as a lower bound, and we will do the same. It is only for small values of g that we see hope of better lower bounds.

We call a vertex v *unbalanced* in a g-edge coloring if there are colors i and j with $d(v, i) > d(v, j) + 1$, and *grossly unbalanced* if there are colors i and j with $d(v, i) > d(v, j) + 2$. A vertex that is unbalanced but not grossly unbalanced contributes at most $\lceil \frac{d(v)}{g} \rceil + 1$ to the objective function. Vertices which are not unbalanced are called *balanced*.

The paper [6] describes how to compute the optimum for MTPS when $g = 2$. We include their method for completeness. We may consider each component separately, so let us assume that G is connected. If G is Eulerian with even number of edges, then following an Eulerian tour and coloring edges with alternate colors gives us an equitable 2-edge coloring of G. If G is Eulerian with an odd number of edges, [6] shows that no equitable 2-edge coloring exists; following the Eulerian tour and using alternate colors results in a coloring with one vertex unbalanced but not grossly unbalanced and a 2-edge coloring with objective $L + 1$. If G is not Eulerian we add extra edges between vertices having odd degree to make it Eulerian, then use alternate colors on the Eulerian tour, starting with an extra edge. The resulting 2-edge coloring has objective L.

Next we describe a local optimization procedure, called *quasibalancing*, that can be used to improve an edge coloring to ensure that no grossly unbalanced vertices exist, without creating unbalanced vertices. Suppose that $d(v, i) > d(v, j) + 2$ for some vertex v. Consider the subgraph induced by colors i and j and let H be the component containing v. Use the $g = 2$ procedure to recolor H such that no vertex of H remains unbalanced in i and j except for possibly v. It is easy to check that this procedure reduces the quantity $\sum_{u \in V} \sum_{1 \le k < l \le g} |d(u, k) - d(u, l)|$ and it does not make balanced vertices unbalanced or create any new grossly unbalanced vertices. The repeated application of the procedure results in a g-edge coloring without grossly unbalanced vertices. At the end, $\max_i d(v, i) \le 1 + \lceil \frac{d(v)}{g} \rceil$, so $\sum_{v \in V} \max_i d(v, i) \le L + n \le 2L$, which shows that we have a 2-approximation algorithm. Pseudocode for one step of quasibalancing (balancing two colors) appears later in this paper (Table 2, in Section 4), when we analyze the running time of our algorithms.

We define the $\overline{MTPS}$ problem to be the restriction of MTPS to instances where every vertex has degree at most g. If one uses L as a lower bound (as we do), $\overline{MTPS}$ is equivalent from approximation point of view: we give a simple reduction (also used in [2]) from MTPS to $\overline{MTPS}$. Starting with an instance G of MTPS we construct an instance G' of $\overline{MTPS}$ as follows. Every vertex $v \in G$ is replaced by $\lceil \frac{d(v)}{g} \rceil$ copies. Edges of G are processed one by one and edge $uv \in E(G)$ is replaced by one edge between some copy of u to some copy of v such that all copies of vertices have degree at most g. Then $L = L' = |V(G')|$. Any g-edge coloring of G' translates back to G without an increase in the objective function.

Berry and Modiano's [3] 2-approximation algorithm works as follows. Given a multigraph G, replace each vertex v by $\lceil 3d(v)/(2(g-1)) \rceil$ copies (their conference version [2] uses $\lceil 3d(v)/2g \rceil$ copies, which is not enough when $d = 800$ and $g = 4$, as pointed out by a referee of this paper!) and distribute the endpoints of edges that had been incident to v evenly among its copies, so that each copy has degree at most $(2/3)g$. Then Shannon's bound [10] gives a proper g-edge-coloring of the modified graph; transferring the colors to edges of G yields $d(v,i) \le 2\lceil d(v)/g \rceil$ for each vertex v and color i. Thus the objective is at most $2L$, as claimed.

Both the 2-approximation obtained by [2,3] and the one obtained by quasibalancing suggest the hardest instances for MTPS are g-regular graphs. Since we are using L as a lower bound, we restrict the rest of this paper to the $\overline{MTPS}$ problem. Note that for $\overline{MTPS}$ instances a vertex is properly colored if and only if it is balanced.

3 The Approximation Ratio

Lemma 1. *Given a graph $G = (V, E)$ of maximum degree g, there is a g-edge coloring with at most $\lfloor \frac{|V|-1}{2} \rfloor$ unbalanced vertices. Such a coloring can be obtained in polynomial time.*

Proof. The proof is by induction on $n + g$, where $n = |V|$. The base cases $n = 1$ and $n = 2$ are trivial. If G is not connected, let $G_1 = (V_1, E_1)$ be one connected component and $G_2 = (V_2, E_2)$ be the remaining graph. Induction gives a g-edge coloring with at most

$$\lfloor \frac{|V_1| - 1}{2} \rfloor + \lfloor \frac{|V_2| - 1}{2} \rfloor \le \lfloor \frac{|V_1| + |V_2| - 1}{2} \rfloor$$

unbalanced vertices. Thus we assume G is connected.

We wish to make use of a maximum matching of G, so we apply the Edmonds-Gallai decomposition theorem to the underlying simple graph of G. The statement of this theorem, as in, for example, [8, p94], is described in the next sentence. In polynomial time, using Edmonds' Maximum Matching

Algorithm [4], we obtain a set $A \subseteq V$ such that $G \setminus A$ has components $B_1, B_2, \dots, B_k, D_1, D_2, \dots D_j$ such that:

- for each $1 \leq i \leq j$ D_i has a perfect matching,
- for each $1 \leq i \leq k$ and each vertex $v \in B_i$, $B_i \setminus \{v\}$ has a perfect matching.
- any maximum matching of G matches the vertices of A to vertices in distinct B_i; moreover, the matchings above can be quickly found and extended to a maximum matching M of G.

If $|V|$ is even and $A = \emptyset$, then M is a perfect matching; we color the edges of M with color g, remove them from G, and apply induction. Any vertex of $G \setminus M$ that is properly edge-colored with $(g-1)$ colors will remains properly edge-colored after adding the edges of M colored g.

If $|V|$ is odd and $A = \emptyset$, then there is a matching M that leaves exactly one vertex (say, v) unmatched. The graph $G \setminus M \setminus \{v\}$ has maximum degree at most $g-1$ and thus, by induction, has a $(g-1)$-edge coloring with at most $\lfloor \frac{|V|-2}{2} \rfloor$ unbalanced vertices. We use the color g for the edges of M. Then for edges of the form uv with $u \in V \setminus \{v\}$, we use colors from $\{1, 2, \dots, g-1\}$ such that, if u is proper in the $(g-1)$-edge coloring, it remains proper; a color for uv is always available since the degree of u in G does not exceed g. The vertex v could also be unbalanced, and so the number of unbalanced vertices in the g-coloring of G is at most $\lfloor \frac{|V|-2}{2} \rfloor + 1 = \lfloor \frac{|V|-1}{2} \rfloor$.

Now we may assume that $A \neq \emptyset$. Select in each B_i a vertex v_i adjacent to some vertex of A to specify M so that M restricted to $B_i \setminus \{v_i\}$ is a perfect matching. Consider $G \setminus M \setminus A$. It has at least $k + j$ components, and maximum degree at most $g - 1$: some vertex degrees drop due to being matched in M, and the remaining vertices are adjacent to vertices in A. Let each B_i have $2b_i + 1$ vertices and each D_i have $2d_i$ vertices. We apply recursion to each component, obtaining a $(g-1)$-edge coloring with at most $\sum_{i=1}^{k} b_i + \sum_{i=1}^{j} (d_i - 1)$ unbalanced vertices. We use color g for the edges of M, and for edges of type uv with $u \notin A$ and $v \in A$ we use colors from $\{1, 2, \dots, g\}$ such that, if u is proper in the $(g-1)$-edge coloring, it remains proper; a color for uv is always available since the degree of u in G does not exceed g. For edges with both endpoints in A, we use arbitrary colors. In the resulting g-edge coloring, the number of unbalanced vertices is at most $|A| + \sum_{i=1}^{k} b_i + \sum_{i=1}^{j} (d_i - 1)$. Note that G has $|A| + \sum_{i=1}^{k} (2b_i + 1) + \sum_{i=1}^{j} 2d_i$ vertices.

Since M is not a perfect matching we have $|A| \leq k - 1$, and therefore

$$\left\lfloor \frac{|A| + \sum_{i=1}^{k} (2b_i + 1) + \sum_{i=1}^{j} 2d_i - 1}{2} \right\rfloor =$$

$$= \left\lfloor \frac{|A| + k - 1}{2} \right\rfloor + \sum_{i=1}^{k} b_i + \sum_{i=1}^{j} d_i \geq |A| + \sum_{i=1}^{k} b_i + \sum_{i=1}^{j} d_i.$$

Thus the number of unbalanced vertices does not exceed $\lfloor \frac{|V|-1}{2} \rfloor$. ■

Using the procedure of the previous lemma followed by quasibalancing we obtain an algorithm that produces a g-edge coloring of a graph of maximum degree g

with at most $n/2$ unbalanced vertices and no grossly unbalanced vertex. The objective is then at most $n+n/2 \leq 3n/2 = 3L/2$, proving the following theorem:

Theorem 1. *There is a* $(3/2)$*-approximation algorithm for MTPS.*

Due to space limitations, we only give the 4/3 proof for $g = 3$. The case $g = 4$ is handled by a more complicated extension of the same method. We use a second local optimization procedure to improve Lemma 1. Quasibalancing is first applied and it assures no grossly unbalanced vertices exists, and one can check it does not make balanced vertices unbalanced.

Consider a vertex v with $d(v,i) = 2$ and $d(v,j) = 0$, and let u_1 and u_2 be the two vertices with vu_1 and vu_2 colored i (u_1 may be u_2). We say that v *can be fixed by* u_1 and v *can be fixed by* u_2. If one of u_1 or u_2, say u_1, is unbalanced (this is also the case when $u_1 = u_2$), then recoloring vu_1 with color j makes v balanced. Then we resume with quasibalancing. If unbalanced vertices v_1 and v_2 can be both fixed by one vertex u, then we fix both v_1 and v_2 Then we resume with quasibalancing.

In all cases, we reduce the number of unbalanced nodes. Thus, at the end of the second local optimization procedure each unbalanced vertex v has two private (not shared with other unbalanced vertices) balanced vertices u_1 and u_2: the two vertices v can be fixed by. This algorithm implies the lemma below for $g = 3$.

Lemma 2. *Assume* $g \leq 4$*. Given a graph* $G = (V, E)$ *of maximum degree* g*, there is a* g*-edge coloring with at most* $\lfloor \frac{|V|}{3} \rfloor$ *unbalanced vertices. Such a coloring can be obtained in polynomial time.*

The lemma is tight: for $g = 2$ consider a triangle $v_1v_2v_3$, while for $g = 3$ or $g = 4$ add one or two parallel edges between v_1 and v_2. Using the procedure of the previous lemma followed by quasibalancing we obtain an algorithm that produces, for $g \leq 4$, a g-edge coloring of a graph of maximum degree g with at most $n/3$ unbalanced vertices and no grossly unbalanced vertex. The objective is then at most $n + n/3 \leq 4n/3 = 4L/3$, and we have a (4/3)-approximation algorithm for MTPS.

4 Implementation and Running Time Analysis

We start with an equivalent version of the algorithm of Lemma 1, given in Table 1. The next paragraph discusses the equivalence.

In the proof of Lemma 1, the third case considered ($A \neq \emptyset$) is always followed by applying to non-trivial components B_j or D_j either the first case ($A = \emptyset$ and $|V(G)|$ even) or the second case ($A = \emptyset$ and $|V(G)|$ odd). Trivial components, (one vertex only, whose only neighbors are in A) give each a proper vertex immediately. When the algorithm, as described in Table 1, encounters the third case, it merges this next application into the same step, removing in Step 5 one vertex from each non-trivial component with odd number of vertices (such a vertex joins A in the set of vertices we gave up on being proper).

Table 1. The algorithm of Lemma 1. The set A above is the set of A of the proof of the lemma, and we mention that each shrunk blossom becomes an outer vertex; thus the final inner vertices are vertices of the original graph. L is designed to have exactly one vertex from each non-trivial odd component of the subgraph of G induced by $V \setminus A$.

Input: Positive integer parameter g and graph $G = (V, E)$ of maximum degree g
Output: Coloring of $E(G)$ with colors $1, 2, \ldots, g$

```
1    If $g = 1$, color all the edges 1 and **return**
2    Compute a maximum matching $M \subseteq E$ in $G$
3    Construct, as in Edmonds' algorithm, alternating forests,
     implicitly shrinking the blossoms
4    Let $Q$ be the set of unmatched vertices, $A$ be the set of inner vertices in the
     alternating forest, and $J$ be the vertices matched by $M$ to some vertex of $A$.
     Let $L$ be the subset of $Q \cup J$ with vertices who have a neighbor outside $A$.
5    Recurse on $G \setminus (A \cup L) \setminus M$ with parameter $g - 1$
6    Assign color $g$ to the edges of $M$
7    **for** each edge $e$ incident to some vertex $u \in A \cup L$
8        **if** $v$, the other endpoint of $e$, is in $V \setminus (A \cup L)$
9            color $e$ such that, if $v$ is proper in $G \setminus A \setminus L$, then $v$ remains proper
10       **else** color $e$ arbitrarily
11   **return**
```

For the running time of the algorithm in Table 1, we first note that g maximum matchings are computed in a graph, which we call $G' = (V', E')$, with at most $n' := n + 2m/g$ vertices and $m' := m$ edges, where n and m are the number of vertices and edges of the original graph. Using the algorithm of Micali and Vazirani [9], we obtain a total running time of $O(gm\sqrt{n + m/g})$. We need to elaborate a bit on steps 7-10. To ensure that proper vertices remain proper, we keep for each vertex v an array M_v of size g indicating which color is already used by edges incident to v. In addition we keep for v an integer j_v (initially 0) such that colors $1, \ldots, j_v$ are used. An unused color for an edge incident to v is found by increasing j_v and testing (and eventually updating) the array M_v. Since j only increases, the total time spent for vertex v on finding unused colors incident to v is $O(g(n + m/g)) = O(gn + m)$.

We move to quasibalancing, which we apply directly to the output of the algorithm in Table 1. Thus the input consists of G', a graph of maximum degree at most g, and whose edges are colored with colors $1, 2, \ldots, g$. Applying quasibalancing to G', rather than the original graph, does not affect the 1.5 approximation ratio and makes the running time easier to analyze.

As described in Section 2, quasibalancing is clearly polynomial: in $O(m + n)$ time we reduce the quantity $\sum_{u \in V} \sum_{1 \leq k < l \leq g} |d(u, k) - d(u, l)|$ by one, and the initial $\sum_{u \in V} \sum_{1 \leq k < l \leq g} |d(u, k) - d(u, l)| \leq ng^3$. We can do a much better analysis if we carefully describe the procedure, changing it a bit and adding randomization. We describe this specific implementation below. But first, some intuition.

The goal, roughly speaking, is to bound the number of times Euler tours are constructed, as in Section 2 or later below. A natural way to obtain such a bound is to have a potential function which decreases fast whenever we do an Euler tour construction and recoloring; each such Euler tour comes from edges colored with only two colors, say i and j. It can reasonably be hoped that picking i and j such as to decrease the potential the most would be good. But to show a good pair of colors exists it is natural to compute the average decrease in potential over pairs i, j. Then we will not even have to pick the best i, j: we save time by picking them randomly. Such an approach, with potential function $\sum_v \sum_i (d(v,i) - 1)^2$ would have worked and give the same bound we give on the running time, provided the recoloring is "perfect" in the sense that all vertices are left balanced in colors i, j. But one vertex v can be left unbalanced in each Euler tour; in particular, when $d(v,i) = 3$ and $d(v,j) = 1$ no progress may be done. There exist graphs on which the "random pair of colors" (and also "best pair of colors") approach fails. So, instead we use randomization in a more complicated way.

Recall that $d(v,j)$ is the number of edges incident to v with color j. Thus $\sum_{1 \le j \le g} d(v,j) \le g$, and $\sum_{j,v} d(v,j) = 2m$. We also think of these $d(v,j)$ as being values in a matrix M where rows are indexed by colors, and columns by vertices. Our goal is to ensure no grossly unbalanced vertex remains, which in our graph of maximum degree bounded by g means that we must reach that $d(v,i) \le 2$ for all $v \in V'$ and $i \in \{1, 2, \ldots, g\}$.

The basic move is to pick two colors and "balance" them. For this an Euler tour is produced; it is important for the analysis that a certain edge is picked as the first edge of the tour, and a color is assigned to this edge. Pseudocode appears in Table 2, with one of the colors being 1. To analyze the total running time of quasibalancing, we give a special role (to be described later) to color 1.

Table 2. Balancing colors 1 and j

```
Let $H$ be the subgraph of $G'$ induced by the edges colored 1 and $j$
Apply the following to each connected component $C$ of $H$
If $C$ is not Eulerian, add "fake" edges between vertices of odd degree
  to obtain graph $C'$
Compute $T$, an Eulerian tour of $C'$
if there is a fake edge
    start $T$ with a fake edge, colored arbitrarily
else
    if there is $v \in V(C)$ with $d(v,1) = 2$ and $d(v,j) = 0$, or $d(v,1) = 0$ and $d(v,j) = 2$
        start $T$ with an edge incident with $v$, colored $j$
    else
        if there is $v \in V(C)$ with $d(v,1) \neq d(v,j)$
            start $T$ with an edge incident with $v$, colored 1
        else
            start $T$ with an arbitrary edge, colored arbitrarily
Follow $T$, assigning colors to its edges alternating 1 and $j$
Remove the fake edge, if any
```

The running time of the procedure is $O(n' + |E(H)|) = O(\sum_{v \in V'}(1 + d(v,1) + d(v,j)))$. The procedure results in $|d(v,1) - d(v,j)| \leq 1$ at every vertex of the component C except perhaps one vertex v (where the Eulerian tour starts and ends), in which case $|d(v,1) - d(v,j)| = 2$. Here we choose which of $d(v,1)$ or $d(v,j)$ is larger: If the two values are 0 and 2, we make $d(v,1) = 0$ and $d(v,j) = 2$; otherwise we make $d(v,1) > d(v,j)$.

For each vertex v, let $||v||_j = [\max\{d(v,j) - 2, 0\}]^2$ and let $||v|| = \sum_j ||v||_j$. $||v||$ is defined to measure, roughly speaking, the progress of a balancing: when $||v|| = 0$, we have $d(v,j) \leq 2$ for all j. More formally, we define the *progress*(v) of a balancing as $||v|| - ||v'||$, where $||v'||$ is $||v||$ after balancing.

Lemma 3. *For any balancing of two rows i and j,* progress(v) *is a nonnegative integer. Moreover, if we balance rows with values $d(v,i)$ and $d(v,j)$ being M, m, with $m \leq M$, and after the balancing $d(v,i)$ and $d(v,j)$ are (not necessarily in this order) M', m', with $m' \leq M'$, then* progress$(v) \geq 1$ *unless $M = m$, $M = m + 1$, $M = M' = m + 2 = m' + 2$, or $M \leq 2$.*

Proof. Clearly *progress*(v) is an integer.

If $M = m + 1$ then there is a "fake" edge added at v, which eventually yields $M' = m' + 1$. If $M = m$ then the Eulerian tour through v either has even length or it does not start at v; this yields $M' = m'$. In each case *progress*$(v) = 0$, so we may assume that $M \geq m + 2$.

If m, m', M, M' are each at least 2, then *progress*$(v) = (m-2)^2 + (M-2)^2 - [(m'-2)^2 + (M'-2)^2]$. According to the algorithm in Table 2 we get $M' \leq m' + 2$. Since $M + m = M' + m'$, the average of the two values does not change. Thus we have $m \leq m' \leq \frac{M+m}{2} \leq M' \leq M$. It follows that $(m-2)^2 + (M-2)^2 - [(m'-2)^2 + (M'-2)^2]$ is nonnegative, with equality if and only if $m = m'$ and $M' = M$.

If $M \geq 4$ and m is 0 or 1, *progress*(v) is at least

$$\begin{aligned}
&(M-2)^2 - [(m'-2)^2 + (M'-2)^2] \\
&\geq (M-2)^2 - [(\frac{M+m}{2} - 3)^2 + (\frac{M+m}{2} - 1)^2] \\
&= (M-2)^2 - [2(\frac{M+m}{2} - 2)^2 + 2] \\
&\geq (M-2)^2 - 2(\frac{M+1}{2} - 2)^2 - 2 \\
&= \frac{1}{2}M^2 - M - \frac{5}{2} \geq 0.
\end{aligned}$$

If $M = 3$ and $M' \leq 2$ then $m, m' \leq 2$ and *progress*$(v) = 1$. Otherwise $M = M'$ or $M \leq 2$ and *progress*$(v) = 0$. ■

Consider the subsequence of rows $r = \sigma(1) < \sigma(2) < \ldots < \sigma(k)$ for which $d(v,r) \neq 1$.

Claim. Suppose that $d(v,\sigma(j)) \geq 3$, $d(v,\sigma(j+1)) = 0$, and either $d(v,\sigma(j-1)) = 0$ or $j = 1$. If $\sigma(j) \neq 1$ then balancing row 1 with rows $\sigma(j-1), \ldots, \sigma(j+1)$

yields *progress*$(v) \geq ||v||_{\sigma(j)}/80$. The same progress is achieved when $\sigma(j) = 1$ by balancing row 1 with rows $2, \ldots, \sigma(2)$.

Proof. Let $y = d(v, \sigma(j))$. If $\sigma(j) > 1$, let b be the value of $d(v, 1)$ just after balancing row $\sigma(j)$ and row 1. Since the value of $d(v, 1)$ before this balancing was nonnegative, $b \geq \frac{y}{2} - 1$. If $\sigma(j) = 1$ (in which case $j = 1$ and $y = d(v, 1)$), then we let $b = y$. Suppose that $b \geq 4$.

¿From the hypothesis, $d(v, \sigma(j + 1)) = 0$, and if $\sigma(j + 1) > \sigma(j) + 1$, then by the definition of the σ's we have $d(v, \sigma(j) + 1) = 1$. Thus, there a 0 or 1 in row $\sigma(j) + 1$, and therefore after balancing rows 1 and $\sigma(j) + 1$, the new entries are either $\frac{b+1}{2} \pm \epsilon$ or $\frac{b}{2} \pm \epsilon$, integers with $\epsilon \in \{0, \frac{1}{2}, 1\}$. Thus $||v||_1 + ||v||_j$ becomes at most $(\frac{b+1}{2} + \epsilon - 2)^2 + (\frac{b+1}{2} - \epsilon - 2)^2$ or $(\frac{b}{2} + \epsilon - 2)^2 + (\frac{b}{2} - \epsilon - 2)^2$. The largest this could be is $(\frac{b+1}{2} - 1)^2 + (\frac{b+1}{2} - 3)^2$, so *progress*$(v)$ is at least

$$\begin{aligned} &(b-2)^2 - [(\frac{b+1}{2} - 1)^2 + (\frac{b+1}{2} - 3)^2] \\ &= (b-2)^2 - [2(\frac{b+1}{2} - 2)^2 - 2] \\ &= \frac{1}{2}b^2 - b - \frac{5}{2}. \end{aligned}$$

Since $\frac{9}{20}b^2 - b - \frac{5}{2} > 0$ when $b \geq 4$, *progress*$(v) \geq \frac{1}{20}b^2 \geq \frac{1}{80}(y-2)^2$, which suffices.

If $b \leq 3$ then $(y-2)^2/80 \leq 36/80 < 1$, so *progress*$(v) \geq 1$ would suffice.

If $\sigma(j) = 1$ (and $j = 1$) then $d(v, 1) \geq 3$ at first, and is balanced only with 1s until row $\sigma(j + 1)$. Each balancing with a 1 that doesn't yield positive progress must begin and end with $d(v, 1) = 3$. Thus, if there has been no progress until row 1 is balanced with row $\sigma(j + 1)$, then at that point 0 is balanced with a 3 (or $d(v, 1) \geq 3$ if $\sigma(j + 1) = \sigma(1) + 1$), for positive progress.

If $j = 1$ and $\sigma(j) \neq 1$, then $d(v, 1)$ is 0 or 1 until row 1 is balanced with row $\sigma(j)$, which yields positive progress or makes $d(v, 1) = 3$. In the latter case, the situation is identical to the previous case, and thus will later yield positive progress.

If $j > 1$ and no progress is made balancing row 1 and row $\sigma(j - 1)$, then $d(v, 1)$ becomes 0 or 1, after which it follows the previous case. ■

Each pass of the algorithm randomly orders the colors (redefining the rows), then balances row 1 with rows $2, \ldots, g$ in sequence. See Table 3.

Lemma 4. *One pass takes $O(n'g)$ time.*

Proof. Recall that balancing rows 1 and j takes time $O(\sum_{v \in V'}(1 + d(v, 1) + d(v, j)))$. Let $t(v, j) = 1 + d(v, 1) + d(v, j)$ and $t(v) = \sum_{j=2}^{g} t(v, j)$; note however that $d(v, 1)$ changes after each balancing. It is enough to show that $t(v) = O(g)$ for all $v \in V'$.

We employ a credit scheme for the proof. Row j starts with $2d(v,j)+3$ credits. We maintain the invariant that row 1 has at least $2d(v,1)$ credits. Consider the rebalancing of row 1 with row j. Let $d'(v,1)$ be the number of edges colored 1 incident to v after the rebalancing. Row j brings $2d(v,j)+3$ credits, and we have $2d(v,1)$ credits in row 1. Of these, $1+d(v,1)+d(v,j)$ go toward $t(v,j)$, and we are left with $d(v,1)+d(v,j)+2$ credits. Since $d'(v,1) \leq 1+(d(v,1)+d(v,j))/2$, the credit invariant is maintained. Thus $t(v) \leq \sum_{j=1}^{g}(3+2d(v,j)) \leq 3g+2g = O(g)$, completing the proof. ■

Table 3. One pass of the algorithm through the colors

```
Randomly reorder colors $1,\ldots,g$, redefining the rows
for $j = 2,\ldots,g$
    balance row 1 and row $j$ according to the algorithm in Table 2
endfor
```

We continue with estimating the total number of passes. Consider any vertex v with $\max_j d(v,j) \geq 3$, and let $X_q = X_q(v)$ be the value of $||v||$ after q passes. Each X_q is a random variable, and X_0 is a constant bounded by the maximum possible value of $||v||$, which is clearly $(g-2)^2$. We begin by showing that $E[X_q|X_0,\ldots,X_{q-1}] \leq (1279/1280)X_{q-1}$.

As above, we consider the subsequence of rows $r = \sigma(1) < \sigma(2) < \ldots < \sigma(k)$ for which $d(v,r) \neq 1$. Since $\sum_{j=1}^{g} d(v,j) \leq g$, at least half of these $d(v,r)$ are 0s. Each color which has degree 3 or more is now in a row $r = \sigma(j)$ for some $1 \leq j \leq k$. We are *happy* with the color in row $\sigma(j)$ if the entry in row $\sigma(j+1)$ is 0 (and $j+1 \leq k$) and the entries in rows $\sigma(j-1)$ and $\sigma(j-2)$ are either 0 or undefined (in case j is 1 or 2). By Claim 4, any happy row $r = \sigma(j)$ with $1 \leq j < k$ yields progress at least $||v||_r/80$ from balancing row 1 with rows $\sigma(j),\ldots,\sigma(j+1)$ (rows $2,\ldots,\sigma(j+1)$ if $\sigma(j)=1$). Observe that there are at least two rows between happy rows in the subsequence $\sigma(1),\ldots,\sigma(k)$, so the progress counted for one happy row does not overlap with progress counted for another happy row.

Suppose that $k \geq 5$. Each color with $d(v,j) \geq 3$ is placed in a row j with $j \neq k$ with probability $\frac{k-1}{k}$. The probability that such a color is happy is at least $(\frac{k/2}{k-1})(\frac{k/2-1}{k-2})(\frac{k/2-2}{k-3}) = \frac{1}{8}\frac{k(k-4)}{(k-1)(k-3)} \geq \frac{1}{16}\frac{k}{k-1}$. Thus the expected progress for one pass at v is at least the sum of $\frac{1}{16}(||v||_r/80)$ over all r such that $d_r(v) \geq 3$. Since $||v||_r = 0$ whenever $d_r(v) \leq 2$, also $\sum_r \frac{1}{1280}||v||_r = \frac{1}{1280}||v||$. Therefore $E[X_q|X_0,\ldots,X_{q-1}] \leq (1279/1280)X_{q-1}$ as desired.

If k is 3 or 4, using $\max_j d(v,j) \geq 3$, it follows that exactly one row of the subsequence has a nonzero entry. Hence the color with $d(v,j) \geq 3$ is happy with probability 1/3 or 1/4. As this is greater than $\frac{1}{16}$, in this case we still have $E[X_q|X_0,\ldots,X_{q-1}] \leq (1279/1280)X_{q-1}$ in this case.

Let Y_q be the value of $\sum_v ||v||$ after q passes. Then Y_q is the sum of $X_q(v)$ over all vertices v for which $\max_j d(v,j) \geq 3$. By linearity of expectation, we have $E[Y_q|Y_0,\ldots,Y_{q-1}] \leq (1279/1280)Y_{q-1}$. It immediately follows that

$E[Y_q] = E[Y_q|Y_0] \leq (1279/1280)^q Y_0$. By Markov's inequality, $Pr(Y_q > 0) < (1279/1280)^q Y_0$. Let $q = 2\log_{1279/1280}(n'(g-2)^2)$. Then $(1279/1280)^q Y_0 \leq (n'(g-2)^2)^{-2+1} < 1/n$, so $Y_q = 0$ with high probability. That is, after $O(\log(n'g))$ passes, $||v||$ is zero w.h.p. Since each pass takes time $O(n'g)$, the total time needed is $O(n'g \log n'g)$.

With $n' \leq n + 2m/g$, the main result of this paper is:

Theorem 2. *There is a randomized $O(gm\sqrt{n+m/g})$ algorithm that gives a* 1.5 *approximation to MTPS.*

We omit for lack of space the running time analysis for $g = 3$ and $g = 4$.

References

1. Sanjeev Arora, Carsten Lund, Rajeev Motwani, Madhu Sudan, and Mario Szegedy. Proof verification and hardness of approximation problems. *Journal of ACM*, 45(3):501–555, 1998.
2. Randall A. Berry and Eytan Modiano. On the benefit of tunability in reducing electronic port counts in WDM/TDM networks. In *INFOCOM*, volume 2, pages 1340–1351, 2004.
3. Randall A. Berry and Eytan Modiano. Optimal transceiver scheduling in WDM/TDM networks. *IEEE Journal on Selected Areas in Communications*, 23(8):1471–1495, 2005.
4. Jack Edmonds. Paths, trees, and flowers. *Canadian J. Math*, 17:449–467, 1965.
5. U. Feige, E. Ofek, and U. Wieder. Approximating maximum edge coloring in multigraphs. In *APPROX, volume 2462 of LNCS*, pages 108–121, 2002.
6. A. J. W. Hilton and D. de Werra. A sufficient condition for equitable edge-colourings of simple graphs. *Discrete Mathematics*, 128:179–201, 1994.
7. Ian Holyer. The NP-completeness of edge-coloring. *SIAM J. Comput.*, 10(4):718–720, 1981.
8. L. Lovász and M. D. Plummer. *Matching Theory.* Elsevier Science, 1986.
9. Silvio Micali and Vijay V. Vazirani. An $O(\sqrt{|V|}|e|)$ algorithm for finding maximum matching in general graphs. In *FOCS*, pages 17–27, 1980.
10. Claude E. Shannon. A theorem on coloring the lines of a network. *J. Math. Phys*, 28:148–151, 1949.

Zero Error List-Decoding Capacity of the $q/(q-1)$ Channel

Sourav Chakraborty[1], Jaikumar Radhakrishnan[2,3],
Nandakumar Raghunathan[4,] [⋆], and Prashant Sasatte[5,] [⋆⋆]

[1] Department of Computer Science,
University of Chicago, Chicago, USA
[2] Toyota Technological Institute at Chicago, USA
[3] School of Technology and Computer Science,
Tata Institute of Fundamental Research, Mumbai, India
[4] Microsoft, Seattle, USA
[5] University of Waterloo, Waterloo, Canada

Abstract. Let m, q, ℓ be positive integers such that $m \geq \ell \geq q$. A family $\mathcal{H}$ of functions from $[m]$ to $[q]$ is said to be an (m, q, ℓ)-family if for every subset S of $[m]$ with ℓ elements, there is an $h \in \mathcal{H}$ such that $h(S) = [q]$. Let, $N(m, q, \ell)$ be the size of the smallest (m, q, ℓ)-family. We show that for all q, $\ell \leq 1.58q$ and all sufficiently large m, we have

$$N(m, q, \ell) = \exp(\Omega(q)) \log m.$$

Special cases of this follow from results shown earlier in the context of perfect hashing: a theorem of Fredman & Komlós (1984) implies that $N(m, q, q) = \exp(\Omega(q)) \log m$, and a theorem of Körner (1986) shows that $N(m, q, q+1) = \exp(\Omega(q)) \log m$. We conjecture that $N(m, q, \ell) = \exp(\Omega(q)) \log m$ if $\ell = O(q)$. A standard probabilistic construction shows that for all q, $\ell \geq q$ and all sufficiently large m,

$$N(m, q, \ell) = \exp(O(q)) \log m.$$

Our motivation for studying this problem arises from its close connection to a problem in coding theory, namely, the problem of determining the *zero error* list-decoding capacity for a certain channel studied by Elias [IEEE Transactions on Information Theory, Vol. 34, No. 5, 1070–1074, 1988]. Our result implies that for the so called $q/(q-1)$ channel, the capacity is exponentially small in q, even if the list size is allowed to be as big as $1.58q$. The earlier results of Fredman & Komlós and Körner, cited above, imply that the capacity is exponentially small if the list size is at most $q+1$.

1 Introduction

Shannon [S56] studied the zero error capacity of discrete finite memoryless noisy channels. Such a channel can be modeled as a bipartite graph (V, W, E), where $(v, w) \in E$

[⋆] Work done while the author was at the University of Chicago.

[⋆⋆] Work done while the author was at TIFR, Mumbai.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 129–138, 2006.

iff the letter w can be received when the letter v is transmitted. The goal then, is to encode messages as strings of letters from the input alphabet V and recover it from the received message. The goal naturally is to use as few input letters as possible and still recover the intended message perfectly. Shannon [S56] and Lovász [L79] determined the best rate of transmission achievable under this model for several specific channels.

We are interested in the list-decoding version of this problem, studied by Elias [E88]. For example, consider the channel shown in Figure 1. It is not hard to see that for this channel, no matter how many letters are used in the encoding, it is impossible to recover an input message uniquely (assuming there are at least two possibilities for the input message). However, it is not hard to see that one can always encode messages using strings of letters such that based on the received message one can narrow down the possibilities to just two, that is, we cannot decode exactly but we can list-decode with a list of size two. This motivates the following definition.

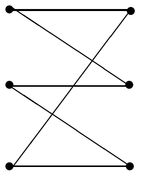


Fig. 1. The 3/2 channel

Definition 1 (Code, Rate). *Consider a channel $C = ([q], [q'], E)$ with q input letters and q' output letters. We say that a sequence $\sigma \in [q]^n$ is compatible with $\sigma' \in [q']^n$ if for $i = 1, 2, \ldots, n$, we have $(\sigma[i], \sigma'[i]) \in E$. A subset $S \subseteq [q]^n$ is said to be a zero error ℓ-list-decoding code for the channel C if for all σ' in $[q']^n$,*

$$|\{\sigma \in S : \sigma \textit{ and } \sigma' \textit{ are compatible}\}| \leq \ell.$$

Let $n(m, C, \ell)$ be the minimum n such that there is a zero error ℓ-list-decoding code S for the channel C, such that $S \geq m$. The zero error list-of-ℓ rate of the code S is

$$R_{C,\ell}(S) = \frac{1}{n} \log \left(\frac{m}{\ell}\right),$$

and the zero error capacity of the channel $\mathcal{C}$ is the least upper bound of the attainable zero error list-of-ℓ rates of all codes.

For the $3/2$ channel Elias [E88] proved that the zero error capacity when $\ell = 2$ is lower bounded by $\log(3) - 1.5 \approx 0.08$ and upper bounded (see (2) below) by $\log(3) - 1 \approx 0.58$. In this paper, we study generalization's of the $3/2$ channel. The $q/(q-1)$ channel corresponds to the complete bipartite graph $K_{q,q}$ minus a perfect matching. Thus, the transmission of any letter can result in all but one of the letters being received. It is easy to see that that it is not possible to design a code where one can always recover the original message exactly. However, it is possible to design codes that perform list-decoding with lists of size $q - 1$. In fact a routine probabilistic argument shows the following.

Proposition 1. $n(m, q, q-1) = \exp(O(q)) \log m$.

The $q/(q-1)$ channel is thus a natural and simple channel where exact decoding is not possible, but list-decoding with moderate size lists is possible. The main point of interest for us is that the rate of the code promised by Proposition 1 is exponentially small as a function of q. Is this exponentially small rate the best we can hope for if the list size is restricted to be $q-1$? Yes, and this follows from a lower bound on the size of families of perfect hash functions shown by Fredman and Komlós [FK84]. A generalization of the result of Fredman and Komlós obtained by Körner [K86], implies that the rate is exponentially small even if we allow the decoder to produce lists of size q. For what list size, then, can we expect list-decoding codes with constant or inverse polynomial rate?

Proposition 2. *For all q we have, $n(m, q, \lceil q \ln q \rceil) = O(q \log m)$.*

On the other hand, it can be shown that the rate cannot be better than $\frac{1}{q}$ unless the list size is allowed to depend on m.

Proposition 3. *All functions $\ell : \mathbb{Z}^+ \to \mathbb{Z}^+$ and all q, for all large enough m, $n(m, q, \ell(q)) \geq q \log m$.*

Thus, we know that the rate is exponentially small when the list size is required to be exactly q, and it is an inverse polynomial when the list size is $\theta(q \ln q)$. These observations, however, do not completely determine the dependence of the rate on the list size, or even the smallest list size (as a function of q) for which there are codes with rate significantly better than an inverse exponential. We conjecture the following.

Conjecture 1. *The conjecture has two parts.*

1. *For all constants $c > 0$, there is a constant ϵ, such that for all large m, we have*

$$n(m, q, cq) \geq \exp(\epsilon q) \log m$$

2. *For all function $\ell(q) = o(q \log q)$ and for all large m we have*

$$n(m, q, \ell(q)) \geq q^{\omega(1)} \log m$$

In this paper, we make progress towards the first part of the conjecture.

Theorem 1 (Main result). *For $\epsilon > 0$, there is a $\delta > 0$ such that for all large q and for all large enough m, we have $n(m, q, (\gamma - \epsilon)q)) \geq \exp(\delta q) \log m$, where $\gamma = e/(e-1) \approx 1.58$.*

2 Techniques

As stated above the inverse exponential upper bounds on the rate when the list size is $q-1$ or q follow from results proved earlier in connection with hashing. In this section, we formally state this connection, review the previous techniques, and then outline the argument we use to obtain our result.

2.1 Connection to Hashing

Definition 2. *Let q, ℓ, m be integers such that $1 \leq q \leq \ell \leq m$. A family $\mathcal{H}$ of functions from $[m]$ to $[q]$ is said to be an (m, q, ℓ)-family of hash functions if for all ℓ-sized subsets S of $[m]$, there is a function $h \in \mathcal{H}$ such that $h(S) = [q]$. $N(m, q, \ell)$ is the size of the smallest (m, q, ℓ)-family of hash functions. For convenience, we will allow m, q and ℓ to positive real numbers, and use $N(m, q, \ell)$ to mean the size of the smallest (m', q', ℓ')-family where m', q' and ℓ' are integers such that $m' \geq m$, $q' \geq q$ and $\ell' \leq \ell$.*

The connection between the family of hash functions and zero error list-decoding codes for the $q/(q-1)$ channel is as follows. Suppose we have an (m, q, ℓ)-code $\mathcal{C} \subseteq [q]^n$. Such a code naturally gives rise to n functions h_1, h_2, ..., h_n from $[m]$ to $[q]$: where $h_i(j) = k$ iff the i-th letter of the j-th codeword is $k \in [q]$. It is then straightforward to verify that for every set $S \subseteq [m]$ of size $\ell + 1$, we have $h_i(S) = [q]$ for some $i \in \{1, 2, \ldots, n\}$. This translations works in the other direction as well: if $\{h_1, h_2, \ldots, h_n\}$ is an (m, q, ℓ)-family, then the code $\mathcal{C} \subseteq [q]^n$ of size m, whose j-th word $w_j \in [q]^n$ $(1 \leq j \leq m)$ is defined by $w_j[i] = h_i(j)$ is an $(m, q, \ell - 1)$-code. We have thus verified the following claim.

Proposition 4. *For all m, q, and ℓ, we have $n(m, q, \ell) = N(m, q, \ell + 1)$.*

In light of the above, we will concentrate on showing lower bounds for $N(m, q, \ell)$. Our main result can then be reformulated as follows.

Theorem 2. *For all $\epsilon > 0$, there is a $\delta > 0$, such that for all large q, $\ell \leq \left(\frac{e}{e-1} - \epsilon\right) q$, and all large enough m,*

$$N(m, q, \ell) \geq \exp(\delta q) \log m.$$

It is easy to see that Theorem 1 follows immediately from this. In Section 4.2 we will formally prove this theorem. We now present an overview.

2.2 The Lower Bound Argument

It will be helpful to review the proof of the lower bound shown by Fredman and Komlós [FK84].

Theorem 3. *For all large q and all large enough m, $N(m, q, q) \geq O(q^{-1/2} \exp(q)) \log m$.*

First, we note two simple lower bounds on $N(m, q, q)$. First, any one hash function can perfectly hash at most $\left(\frac{m}{q}\right)^q$ sets of size q. So,

$$N(m, q, \ell) \geq \binom{m}{q} \left(\frac{m}{q}\right)^{-q} \approx \frac{1}{\sqrt{2\pi q}} \exp(q). \tag{1}$$

This bound has the required exponential dependence on q but not the logarithmic dependence on m. A different argument gives us a logarithmic dependence on m. If we restrict attention to all elements of $[m]$ that are mapped by the first hash function to some $q - 1$

of the $[q]$ elements of $[m]$, then clearly every q-sized subset of these elements must be perfectly hashed by at least one of the remaining hash functions. From this, we conclude that $N(m,q,q) \geq N\left((\frac{q-1}{q})m, q, q\right)$ (provided $m \geq q$), which implies

$$N(m,q,q) \geq \log_{\frac{q}{q-1}}\left(\frac{m}{q-1}\right) \geq q \log\left(\frac{m}{q-1}\right). \tag{2}$$

[A similar calculation can be used to justify Proposition 3.] This bound, gives us the required logarithmic dependence on m but not the exponential dependence on q. Fredman and Komlós devised an ingenious argument that combined the merits of (1) and (2). Consider a set T of size $q-2$. Clearly, if a function maps two of the elements of T to the same value in $[q]$, then this hash function is incapable of perfectly hashing any q-element superset $T' \supseteq T$. An averaging argument shows that for if T is chosen uniformly at random then all but an exponentially small fraction of the original family do map some two elements of T to the same element. Furthermore, for every two elements of $[m]-T$ one of the remaining hash functions (that are on-to-one on T) must map these two elements differently. By (2), the number of hash functions remaining must be at least $\log(m-q+2)$. Thus, the size of original family must be at $\exp(\Omega(q))\log(m-q+2)$. (The arguments used by Fredman and Komlós and Körner are more sophisticated and yield slightly better bounds.)

Our argument is similar. Suppose we have an (m,q,ℓ) family of hash functions where $\ell = 1+\frac{1}{e}-\epsilon$ (for some $\epsilon > 0$). As in the argument above, we pick a set T of size $q-2$. The main observation now is that for any fixed function $h:[m]\to[q]$, the expected size $h(T)$ (as T is chosen at random) is about $q\left(1-\frac{1}{e}\right)$, and there is a sharp concentration of measure near this mean value. Thus, only for an exponentially small fraction of the hash functions in the family is the image of T at least $q\left(1-\frac{1}{e}+\epsilon\right)$. The majority of the functions already suffer so many collisions on T, that they cannot cover all of $[q]$ when an additional $\left(\frac{1}{e}-\epsilon\right)q$ elements are added to T. Using an argument similar to the one used to show (2), we conclude that an exponentially small fraction of the original family must be at least $\log(m-q+2)$. The lower bound will follow from this. This argument is presented in detail in Section 4.1. It however is somewhat weaker than the bound claimed in Theorem 2. The stronger result is obtained by applying this idea recursively. The formal proof proceeds by induction, and is presented in Section 4.2.

3 Preliminaries

In this section we develop the tools that will be necessary in the proof of Theorem 2 in Section 4.1 and 4.2.

Definition 3 (Derived function). *Let m, q , q' be integers such that $m \geq q > q' \geq 1$ and let $T \subseteq [m]$ Let $h:[m]\to[q]$ be a hash function such that $|h(T)| \leq q-q'$. Then, the function $h_{T,q'}$ is defined as follows. Let $j_1, j_2, \ldots, j_{q'}$ be the smallest q' elements of $[q]-h(T)$. Then, for all $i \in [m]$, let*

$$h_{T,q'}(i) = \begin{cases} k \textit{ if } h(i) = j_k \\ 1 \textit{ otherwise} \end{cases}.$$

The following proposition follows immediately from our definition.

Proposition 5. *Let $h : [m] \to [q]$. If $T, T' \subseteq [m]$ are such that $|h(T)| \leq q - q'$ and $h(T \cup T') = [q]$, then $h_{T,q'}(T') = [q']$.*

Lemma 1. *If $\mathcal{H}$ is a family of hash functions from $[m]$ to $[2]$. Then, there is a subset $U \subseteq [m]$ of size at least $m/2^{|\mathcal{H}|}$ such that $|h(U)| = 1$, for all $h \in \mathcal{H}$.*

Proof. Consider the map from $[m]$ to $\{1,2\}^{|\mathcal{H}|}$ defined by $i \mapsto \langle h(i) : h \in \mathcal{H} \rangle$. The range of this map has size exactly $2^{|\mathcal{H}|}$. It follows that there are at least $m/2^{|\mathcal{H}|}$ elements of the domain $[m]$ that map to the same element. □

Definition 4 (Dangerous function). *We say that the function $h : [m] \to [q]$ is ϵ-dangerous for the set $T \subseteq [m]$ if $|h(T)| \geq q\left(1 - \frac{1}{e} + \epsilon\right)$.*

Lemma 2. *Let $h : [m] \to [q]$. Let T be a random subset of $[m]$ chosen uniformly from among all subsets of $[m]$ of size $q - 2$. Then, if $m \gg q$,*

$$\Pr_T[h \textit{ is } \epsilon\textit{-dangerous for } T] \leq 2\exp(-2\epsilon^2 q).$$

To prove Lemma 2, we will need the following concentration result due to McDiarmid.

Lemma 3 (see McDiarmid [M89]). *Let $X_1, X_2, \ldots, X_n$ be independent random variables with each X_k taking values in a finite set A and let $f : A^n \to \mathbb{R}$. For all k, let f change by at most c_k if only the value of X_k is changed, that is, $max_{x \in A^k}|f(x) - f(y)| \leq c_k$, when x and y differ only in the kth coordinate. If $Y = f(X_1, X_2, \ldots, X_n)$ is the random variable with expectation $\mathbf{E}[Y]$, then for any $t \geq 0$,*

$$\Pr[Y - \mathbf{E}[Y] \geq t] \leq \exp\left(\frac{-2t^2}{\sum_{i=1}^n c_k^2}\right).$$

Proof (of Lemma 2). Pick $q - 2$ elements from $[m]$ with replacement, let the resulting set be T. With probability more than $\left(1 - \frac{\binom{q-2}{2}}{m}\right)$ we have that $|T| = q - 2$. Now, fix an $h \in \mathcal{H}$. For $j \in [q]$, the probability that $j \notin h(T)$ is exactly, $\left(1 - \frac{|h^{-1}(j)|}{m}\right)^{q-2}$. Thus, by linearity of expectation, we have

$$\begin{aligned}
\mathbf{E}[|[q] - h(T)|] &= \sum_{j=1}^{q}\left(1 - \frac{|h^{-1}(j)|}{m}\right)^{q-2} \\
&\geq q\left(1 - \frac{1}{qm}\sum_{j=1}^{q}|h^{-1}(j)|\right)^{q-2} \\
&= q\left(1 - \frac{1}{q}\right)^{q-2} \\
&= q\left(1 + \frac{1}{q-1}\right)^{-(q-2)} \\
&\geq q\exp\left(-\frac{q-2}{q-1}\right) \\
&\geq \frac{q}{e}.
\end{aligned}$$

The second inequality follows from Jensen's inequality. Thus, $\mathbf{E}[|h(T)|] \leq q(1-\frac{1}{e})$. We think of $|h(T)|$ as a function of $f(X_1, X_2, \ldots, X_q)$, of $q-2$ independent random variables $X_1, X_2, \ldots, X_{q-2}$ (each distributed uniformly over the set $[m]$).

Note that f changes at most by 1 if the value of any of the the variables is changed (leaving the rest unchanged). We may thus conclude from Lemma 3 that

$$\Pr\left[|h(T)| \geq q\left(1-\frac{1}{e}+\epsilon\right)\right] \leq \exp\left(-2\frac{\epsilon^2 q^2}{q-2}\right) \leq \exp\left(-2\epsilon^2 q\right).$$

This implies that if T is chosen to be a random set of size $q-2$, then the probability that h is dangerous for T is at most

$$\left(1-\frac{\binom{q-2}{2}}{m}\right)^{-1} \exp\left(-2\epsilon^2 q\right) \leq 2\exp\left(-2\epsilon^2 q\right).$$

□

Corollary 1. *Let $\mathcal{H}$ be a family of hash functions from $[m]$ to $[q]$. Then, there is a set $T \subseteq [m]$ of size $q-2$ such that at most $2\exp(-2\epsilon^2 q)|\mathcal{H}|$ hash functions in $\mathcal{H}$ are ϵ-dangerous for T.*

Proof. Pick T at random. By Lemma 2, the expected number of ϵ-dangerous hash functions for T is at most $2\exp(-2\epsilon^2 q)|\mathcal{H}|$. There must be a at least one choice for T with this property. □

4 Proof of Theorem 2

4.1 A Weaker Bound

Our goal in this section is to show the following weaker form of the main theorem, which will serve as the basis for the inductive argument, when we present the proof of the main result.

Theorem 4. *For $\epsilon > 0$, large q and all large enough m, we have $N(m,q,(\gamma-\epsilon)q) \geq \exp(\delta q)\log m$, where $\gamma = 1+\frac{1}{e} \approx 1.37$.*

Proof. Let $\mathcal{H}$ be an (m,q,ℓ)-family with $\ell \leq (\gamma-\epsilon)q$. By Corollary 1, we have a set T of size $q-2$ such that the number of functions that are ϵ-dangerous for T is at most $2\exp(-2\epsilon^2 q)|\mathcal{H}|$. Fix such a T and consider the derived family

$$\mathcal{H}' = \{h_{T,2} : h \in H \text{ is } \epsilon\text{-dangerous for } T\}.$$

By Lemma 1, there is a set $U \subseteq [m]$ of size $m/2^{|\mathcal{H}'|}$ such that $|h'(U)| = 1$ for all $h' \in \mathcal{H}'$. We claim that $|U| < \left\lceil q\left(\frac{1}{e}-\epsilon\right)\right\rceil$. For, otherwise let T' be a subset of U of size $q' = \left\lceil q\left(\frac{1}{e}-\epsilon\right)\right\rceil$, and consider the set $T \cup T'$. If $h \in \mathcal{H}$ is not dangerous then $|h(T)| < q-q'$ and, therefore, $|h(T \cup T')| < q$. On the other hand if h is ϵ-dangerous for T, then our definition of U ensures that $|h(T')| = 1$ and, therefore, $|h(T \cup T')| \leq |T|+1 < q$. Thus,

$$\frac{m}{2^{|\mathcal{H}'|}} < \left\lceil q\left(\frac{1}{e}-\epsilon\right)\right\rceil.$$

This, together with $|\mathcal{H}'| \leq 2\exp(-2\epsilon^2 q)|\mathcal{H}|$ implies our claim. □

4.2 The General Bound

In this section, we will prove the Theorem 2. It will be convenient to restate it in a form suitable for an inductive proof. For $k \geq 1$ and $\epsilon > 0$, let

$$\ell_k(q, \epsilon) \;=\; q\left(1 + \frac{1}{e} + \frac{1}{e^2} + \cdots + \frac{1}{e^k} - \epsilon\right) - 2k.$$

Theorem 5 (Version of main theorem). *For all $k \geq 1$, $\epsilon > 0$, $q \geq 2k$ and all large enough m,*

$$N(m, q, \ell_k(q, \epsilon)) \geq \frac{1}{4k} \exp\left(\frac{2\epsilon^2 q}{e^{2k}}\right) \log m.$$

Proof. We will use induction on k. The base case $k = 1$, follows from the Theorem 4 proved in the previous section.

Induction step: Suppose the claim is false, that is, there is an (m_k, q_k, ℓ_k)-family $\mathcal{H}_k$ such that $\ell_k \leq \ell_k(q_k, \epsilon)$ and

$$|\mathcal{H}_k| < \frac{1}{4k} \exp\left(\frac{2\epsilon^2 q_k}{e^{2k}}\right) \log m. \tag{3}$$

[We use k in the subscript for parameters associated with the $\mathcal{H}_k$ to make emphasize the correspondence with the parameter k used in the induction.] From $\mathcal{H}_k$ we will derive an $(m_{k-1}, q_{k-1}, \ell_{k-1})$-family $\mathcal{H}_{k-1}$ such that

$$|\mathcal{H}_{k-1}| \leq |\mathcal{H}_k|; \tag{4}$$

$$m_{k-1} \geq m_k^{1-\frac{1}{k}}; \tag{5}$$

$$q_{k-1} \geq q_k\left(\frac{1}{e} - \frac{\epsilon}{4}\right) \geq \frac{q_k}{\epsilon^2}; \tag{6}$$

$$\ell_{k-1} \leq \ell_{k-1}(q_{k-1}, \epsilon). \tag{7}$$

Then, using the induction hypothesis, we obtain

$$\begin{aligned}
|\mathcal{H}_k| \geq |\mathcal{H}_{k-1}| &\geq \frac{1}{4(k-1)} \exp\left(\frac{2\epsilon^2 q_{k-1}}{e^{2(k-1)}}\right) \log m_{k-1} \qquad \text{(by the induction hypothesis)}\\
&\geq \frac{1}{4(k-1)} \exp\left(\frac{2\epsilon^2 q_k}{e^{2k}}\right) \left(1 - \frac{1}{k}\right) \log m_k;\\
&= \frac{1}{4k} \exp\left(\frac{2\epsilon^2 q_k}{e^{2k}}\right) \log m_k,
\end{aligned}$$

contradicting (3).

It remains to describe how $\mathcal{H}_{k-1}$ is obtained from $\mathcal{H}_k$. The idea, as outlined in the introduction, is this. In the hope of using induction, we will first pick a subset T of size $q_k - 2$, which most hash functions map into a small number of elements. These functions can now be viewed as mapping $[m_k] - T$ to $[q_k]$, so that the problem reduces to one of covering a large subset of $[q_k]$ with $\ell_k - q_k + 2$. However, not all functions are guaranteed to be so well-behaved. For the few functions that do perform well on T, we need to take evasive action, by restricting attention to a subset of the universe on which these functions are guaranteed to be fail.

This idea is implemented as follows. Using Corollary 1, we first obtain a set $T \subseteq [m_k]$ of size $q_k - 2$ such that at most

$$2\exp\left(-2\left(\frac{\epsilon}{4}\right)^2 q_k\right)|\mathcal{H}_k| \leq 2\exp\left(-2\left(\frac{\epsilon}{4}\right)^2 q_k\right)\cdot\frac{1}{4k}\exp\left(\frac{2\epsilon^2 q_k}{e^{2k}}\right)\log m \leq \frac{1}{2k}\log m_k$$

hash functions in $\mathcal{H}_k$ are $\left(\frac{\epsilon}{4}\right)$-dangerous for T. Now consider the family of derived functions

$$\mathcal{H}' = \{h_{T,2} : h \in \mathcal{H}_k \text{ is } \left(\tfrac{\epsilon}{4}\right)\text{-dangerous for } T\}.$$

Using Lemma 1, we obtain a set $U \subseteq [m_k]$ of size at least $(m_k - q_k + 2)m_k^{-\frac{1}{2k}}$ such that $|h(U)| = 1$ for all $h \in \mathcal{H}'$. Our family $\mathcal{H}_{k-1}$ will be the following set of hash functions from U to $[q_{k-1}]$ (where $q_{k-1} = \lceil q_k(\frac{1}{e} - \frac{\epsilon}{4})\rceil$).

$$\mathcal{H}_{k-1} = \{h_{T,q_{k-1}} : h \in \mathcal{H}_k \text{ is } \underline{\text{not}} \left(\tfrac{\epsilon}{4}\right)\text{-dangerous for } T\}.$$

We claim that for all $T' \subseteq U$ of size $\ell_k - (q_k - 2)$, there is a function $h \in \mathcal{H}_{k-1}$ such that $h(T') = [q_{k-1}]$. For, consider the set $T \cup T'$ of size ℓ_k. By the definition of $\mathcal{H}_k$ there is an $h \in \mathcal{H}_k$ such that $h(T \cup T') = [q_k]$. Such an h is not $\left(\frac{\epsilon}{4}\right)$-dangerous for T because our definition of U ensures that $|h(T \cup T')| < q_k$. So for such an h we have

$$|h(T)| < q_k\left(1 - \frac{1}{e} + \frac{\epsilon}{4}\right) \leq q_k - \left\lceil q_k(\frac{1}{e} - \frac{\epsilon}{4})\right\rceil = q_k - q_{k-1}.$$

Hence, for such an h, by Proposition 5, we have $h_{T,q_{k-1}}(T') = [q_{k-1}]$.

Thus, $\mathcal{H}_{k-1}$ is an $(m_{k-1}, q_{k-1}, \ell_{k-1})$-family for $\ell_{k-1} = \ell_k - (q_k - 2)$. In particular $l_{k-1} \geq q_{k-1}$. We need to verify (4)–(7). The definition of $|\mathcal{H}_{k-1}|$ immediately implies (4). To verify (5) note that for $m_k \gg q_k$,

$$|U| \geq (m_k - q_k + 2)m_k^{-\frac{1}{2k}} \geq m_k^{1-\frac{1}{k}}.$$

Since $q_{k-1} = \lceil q_k(\frac{1}{e} - \frac{\epsilon}{4})\rceil$, (6) holds. Finally, to justify (7), note that

$$\begin{aligned}\ell_{k-1}(q_{k-1},\epsilon) &\geq q_k\left(\frac{1}{e} - \frac{\epsilon}{4}\right)\left(1 + \frac{1}{e} + \cdots + \frac{1}{e^{k-1}} - \epsilon\right) - 2(k-1)\\ &\geq q_k\left[\frac{1}{e} + \frac{1}{e^2} + \cdots + \frac{1}{e^k} - \frac{\epsilon}{e} - \epsilon\left(1 + \frac{1}{e} + \cdots + \frac{1}{e^{k-1}}\right)\right] - 2(k-1)\\ &\geq q_k\left(\frac{1}{e} + \frac{1}{e^2} + \cdots + \frac{1}{e^k} - \epsilon\right) - 2(k-1)\\ &\geq q_k\left(1 + \frac{1}{e} + \cdots + \frac{1}{e^{k-1}} - \epsilon\right) - 2k - q_k + 2\\ &\geq \ell_k(q_k,\epsilon) - (q_k - 2)\\ &\geq \ell_k - (q_k - 2)\\ &= \ell_{k-1}.\end{aligned}$$

□

Acknowledgement

We thank Venkat Guruswami for introducing us to this problem. We thank the referees for their comments.

References

E88. P. Elias. Zero Error Capacity Under List Decoding, IEEE Transactions on Information Theory, vol. 34, No. 5, (1988): 1070-1074.

FK84. M. Fredman, J. Komlós. On the Size of Separating Systems and Families of Perfect Hash Functions, SIAM J. Alg. and Disc. Meth., Vol. 5, No. 1 (1984): 61-68.

K86. J. Körner. Fredman-Komlós bounds and information theory, SIAM J. Algebraic and Discrete Methods, 7 (1986): 560-570.

L79. L. Lovász. On the Shannon Capacity of a Graph, IEEE Trans. Inform. Theory, Vol. IT-25 (1979): 1-7.

M89. C. McDiarmid. On the method of bounded differences, Surveys in Combinatorics, vol 141 LMS Lecture Notes Series (1989): 148–188.

S56. C.E. Shannon. The zero error capacity of a noisy channel, IEEE Trans. Inform. Theory, Vol. IT-2, no. 3, (1956): 8-19. (Reprinted in D. Slepian, Ed., Key Papers in the Development of Information Theory. New York: IEEE Press (1974): 112-123)

Fast Exponential Algorithms for Maximum r-Regular Induced Subgraph Problems

Sushmita Gupta[1], Venkatesh Raman[2], and Saket Saurabh[2]

[1] Department of Computer Science, Simon Fraser University, Canada
gupta@cs.sfu.ca
[2] The Institute of Mathematical Sciences, Chennai 600 113, India
{vraman, saket}@imsc.res.in

Abstract. Given a graph $G=(V,E)$ on n vertices, the MAXIMUM r-REGULAR INDUCED SUBGRAPH (M-r-RIS) problems ask for a maximum sized subset of vertices $R \subseteq V$ such that the induced subgraph on R, $G[R]$, is r-regular. We give an $\mathcal{O}(c^n)$ time algorithm for these problems for any fixed constant r, where c is a positive constant strictly less than 2, solving a well known open problem. These algorithms are then generalized to solve *counting* and *enumeration* version of these problems in the same time. An interesting consequence of the enumeration algorithm is, that it shows that the number of maximal r-regular induced subgraphs for a fixed constant r on any graph on n vertices is upper bounded by $o(2^n)$.

We then give combinatorial lower bounds on the number of *maximal* r-regular induced subgraphs possible on a graph on n vertices and also give matching *algorithmic* upper bounds.

We use the techniques and results obtained in the paper to obtain an improved exact algorithm for a special case of INDUCED SUBGRAPH ISOMORPHISM that is INDUCED r-REGULAR SUBGRAPH ISOMORPHISM, where r is a constant.

All the algorithms in the paper are simple but their analyses are not. Some of the upper bound proofs or algorithms require a *new and different measure* than the usual number of vertices or edges to measure the progress of the algorithm, and require solving an interesting system of polynomials.

1 Introduction

The problem of finding a MAXIMUM/MINIMUM INDUCED SUBGRAPH having properties like acyclicity [6,14], bipartiteness [3,13], regularity [4,5,7,15,16] and regularity with dominance [2] is among the fundamental problems in graph algorithms. Here we study one such problem, namely the MAXIMUM r-REGULAR INDUCED SUBGRAPH problem. The problem is defined as follows:

> MAXIMUM r-REGULAR INDUCED SUBGRAPH (M-r-RIS): Given an undirected graph $G=(V,E)$, find a maximum subset of vertices $R \subseteq V$ such that the induced subgraph on R, $G[R]$, is r-regular.

When r is 0 or 1, it corresponds to the well studied MAXIMUM INDEPENDENT SET and MAXIMUM INDUCED MATCHING problems respectively. While MAXIMUM INDEPENDENT SET problem is among the six classical NP-complete problems [9], MAXIMUM INDUCED MATCHING problem was introduced by Stockmeyer and Vazirani in

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 139–151, 2006.

[17] who showed it to be NP-complete [17]. But only recently, has it been shown [5] that the problem of finding a maximum sized r-regular induced subgraph is NP-complete for any value of r.

In this paper we look at the M-r-RIS problems (a) from *exact exponential time algorithm* paradigm and (b) from the view point of *combinatorial bounds* on the number of maximal r-regular induced subgraphs possible on a graph on n vertices.

An exact algorithm to find a MAXIMUM INDEPENDENT SET or M-0-RIS problem has attracted a lot of attention in the area of exact exponential time algorithms [7,15] and the current fastest known exact algorithm runs in time $\mathcal{O}(1.2108^n)$ [1][15]. There is no algorithm better than $\Theta(2^n)$ is known for larger values of r.

Here, we give a *simple-generic* algorithm for MAXIMUM r-REGULAR INDUCED SUBGRAPH problems taking $\mathcal{O}(c^n)$ time, $c < 2$, a constant, depending on r alone. As a corollary, we obtain $\mathcal{O}(1.6957^n)$, $\mathcal{O}(1.7069^n)$ and $\mathcal{O}(1.7362^n)$ time algorithms for MAXIMUM INDUCED MATCHING, MAXIMUM 2-REGULAR INDUCED SUBGRAPH and MAXIMUM INDUCED CUBIC SUBGRAPH problems respectively. We then generalize the algorithm to solve the counting and enumeration version of M-r-RIS problems in the same time.

Another interesting consequence of the algorithm is that it gives an algorithmic upper bound of $o(2^n)$ on the number of maximal r-regular induced subgraphs on n vertices, if r is some constant. We then investigate the lower bounds on the number of maximal r-regular induced subgraphs of a graph and observe that for larger values of r, the lower bounds and the upper bounds (mentioned above) on the number of maximal r-regular induced subgraphs on n vertices are "almost identical". For small values of r, we improve the upper bounds using a different technique and give a matching *lower* and *upper* bounds on the number of maximal r-regular induced subgraphs. This generalizes the result of Moon and Moser [12] who showed an upper and lower bound of $3^{n/3}$ on the number of maximal independent sets on a graph on n vertices.

Applications of the algorithms developed in this paper include non trivial exact algorithms for a special case of INDUCED SUBGRAPH ISOMORPHISM problem, that is INDUCED r-REGULAR SUBGRAPH ISOMORPHISM problem, where r is a constant, δ-SEPARATING MAXIMUM MATCHING problem [17] and EFFICIENT EDGE DOMINATING SET problem [10].

All our algorithms are simple but their analyses are non trivial. These algorithms are based on one of the most important and widely used tool of exact algorithms, namely the *Branch & Reduce* paradigm. In this paradigm we obtain an optimal solution to a problem by combining solutions to many subproblems of smaller size. We also use a *new measure* not just the number of vertices or edges to measure the progress of the algorithms and use it extensively in many of our upper bound proofs. Measure other than the number of vertices has been source of many recently developed non trivial exact algorithms [6,7,14]. See recent surveys by Woeginger [18] and Fomin et al. [8] for an overview and recent developments in designing exponential time exact algorithms.

Organization of the Rest of the Paper: In Section 2, we give a *generic* algorithm for MAXIMUM r-REGULAR INDUCED SUBGRAPH problems and then generalize it

[1] We round the base of the exponent in all our algorithms which allows us to ignore polynomial terms and write $O(c^n n^{O(1)})$ as $\mathcal{O}(c^n)$.

to solve the counting and enumeration version of the problems. In Section 3 we give matching *lower* and *upper* bounds on the number of maximal r-regular induced subgraphs for various values of r. In Section 4 we conjure all that we develop that far to give faster exact algorithms for M-r-RIS problems for $r = 1$ and 2 than that is possible from the general theorem. We also obtain a non trivial exact algorithm for INDUCED r-REGULAR SUBGRAPH ISOMORPHISM problem in this section. We conclude with some remarks and open problems in Section 5.

In the rest of the paper, we assume that all our graphs are simple and undirected. Given a graph $G = (V, E)$, n represents the number of vertices, and m represents the number of edges. For a subset $V' \subseteq V$, by $G[V']$ we mean the subgraph of G induced on V'. By $N(u)$ we represent all vertices (excluding u) that are adjacent to u, and by $N[u]$, we refer to $N(u) \cup \{u\}$. Similarly, for a subset $D \subseteq V$, we define $N[D] = \cup_{v \in D} N[v]$.

2 Maximum r-Regular Induced Subgraph

Our algorithm is based on the Branch & Reduce paradigm. It selects a vertex v and on one branch of recursion grows a maximum r-regular induced subgraph without v and on the other a maximum r-regular induced subgraph containing v and then outputs the one with the maximum size. At any point of time in our algorithm we maintain a set R (of possible vertices of a M-r-RIS) and construct one *connected component* of this R. Once we finish one connected component, say R_i, we remove all the neighbors of vertices of R_i which are not in R_i, that is $N[R_i] - R_i$, from the graph and then proceed. Based on the structure of $G[R]$, we divide our algorithm into two phases:

1. ACTIVE PHASE : $G[R]$ is $\emptyset$ or a r regular induced subgraph.
2. GROWTH PHASE : There exists a unique component R_i of $G[R]$ such that $G[R_i]$ is not a r regular subgraph.

In ACTIVE PHASE we initiate constructing a new connected component for the possible M-r-RIS. We select a vertex v and at one branch construct a solution not including v and at other branches we construct a solution containing v and a r-subset of $N(v)$. This leads to $\binom{|N(v)|}{r} + 1$ way branching. In the GROWTH PHASE, we choose a vertex v of an unique component R_i of $G[R]$ ($G[R_i]$ is not a r regular subgraph) such that degree of v in $G[R_i]$ is $r_v < r$ and branch on all possible subsets of size $r - r_v$ of $N(v) - R$, which leads to $\binom{|N(v)-R|}{r-r_v}$ way branching.

At any point of time, our algorithm has a 4 tuple $(G' = (V', E'), G, r, R)$. Here, G' contains the unexplored vertices (vertices which are neither in R nor those which have been removed from the consideration). G is the initial *input* graph. This graph never changes during recursion and is only used for checking whether or not $G[R]$ is induced r-regular. R is a set of vertices already chosen for a possible maximum r-regular induced subgraph. We return $-\infty$ if we detect that the corresponding branch can not lead to a r-regular induced subgraph; for an example if in GROWTH PHASE, we find a vertex $v \in R$ having degree r_v in $G[R]$ but strictly less than $r - r_v$ neighbors in V'. In our algorithm until we state otherwise $N(v)$ and $N[v]$ mean $N_G(v)$ and $N_G[v]$ respectively. The details of our algorithm are presented in Figure 1.

```
Algorithm Max-$r$-RIS ($G' = (V', E'), G, r, R$)

Step 1: [active phase] If $G[R]$ is not $r$ regular and not empty then go to Step 2.
    Step 1a: Obtain a new $G'$ by removing $N[R]$ from $G'$.
    Step 1b: Remove all vertices of degree $< r$ recursively from $G'$.
    Step 1c: If $G'$ is non empty then select a vertex $v$ of maximum degree $d \geq r$ and branch
        in following ways: $(1)$ $v \notin R$, and $(2)$ $v \in R$ and then some $r$ neighbors of $v$ are
        in $R$.
        1. $R_1 \leftarrow$ Max-r-RIS($G' - v, G, r, R$)
        2. for ($S \subseteq N_{G'}(v)$ & $|S| = r$),
           $R_S \leftarrow$ Max-r-RIS($G' - N_{G'}[v], G, r, R \cup S \cup \{v\}$).
        return the set (or number) of maximum size between
                $\{R_1\}$ and $\{R_S \mid S' \subseteq N_{G'}(v)\ |S'| = r\}$.
Step 2: [growth phase] Let $R'$ be the unique component of $G[R]$ such that $G[R']$ is
    not a $r$ regular induced subgraph. $R_1 \leftarrow -\infty$. Choose a vertex $v$ with degree say $r_i$ in
    $G[R']$ such that $1 \leq r_i \leq r - 1$ and $|N(v) \cap V'| \geq r - r_i$.
    1. for ($S \subseteq (N(v) \cap V')$ & $|S| = r - r_i$ & maximum degree of $G[R' \cup S]$ is $\leq r$ )
       $R_S \leftarrow$Max-r-RIS($G' - (N(v) \cap V'), G, r, R \cup S$)
    return the set (or number) of maximum size between
            $\{R_1\}$ and $\{R_{S'} \mid S' \subseteq (N(v) \cap V')$ & $|S'| = r - r_i\}$.
```

Fig. 1. A Generic Algorithm to find a Maximum r-Regular Induced Subgraph

Theorem 1. *Let $G = (V, E)$ be a graph on n vertices and r be a fixed constant. Then there exists a constant c, $c < 2$ such that the* MAXIMUM r-REGULAR INDUCED SUBGRAPH *problem can be solved in $\mathcal{O}(c^n)$ time.*

Proof. The correctness of the algorithm is clear. The analysis of time complexity is involved and we present the details here.

From now onwards let r be a fixed positive constant. Observe that the above algorithm is guided by the following recurrences:

$$T(n) \leq T(n-1) + \binom{d}{r} T(n-d-1) \quad d \geq r \qquad \text{[Active Phase]}. \quad (1)$$

$$T(n) \leq \binom{d}{t} T(n-d) \qquad d \geq t,\ 1 \leq t \leq r-1 \text{ [Growth Phase]}. \quad (2)$$

The smallest positive roots of the following inequalities,

$$h_d(x, r) = x^{d+1} - x^d - \binom{d}{r} \geq 0\ , d \geq r \textbf{ and } g_d(x, t) = x^d - \binom{d}{t} \geq 0\ , d \geq t, 1 \leq t \leq r-1,$$

are solutions to the above recurrences. It is clear that $x = 2$ satisfies these inequalities. Now we show that if r is a constant then we can find a c, a function of r alone, and $c < 2$ satisfying these set of inequalities. We need the following easy lemma for our proof.

Lemma 1. *For any* $r \geq 5$, $\binom{2r}{r} \leq \frac{2^{2r}}{4}$.

We concentrate on the polynomials coming from the ACTIVE PHASE as they represent the dominating recurrences. Observe that

$$x^d - \binom{d}{r} \geq x^d(x-1) - \binom{d}{r} \geq x^{d+1} - x^d - \binom{d}{r}.$$

The inequality holds as $x \leq 2$. This shows that if there exists $c = f(r)$ such that $h_d(f(r), r) \geq 0$ then $g_d(f(r), r) \geq 0$.

Now we show that if there exists a $c = f(r)$ such that $h_{2r}(c, r) \geq 0$ then we can choose a c' such that $h_d(c', r) \geq 0$ for any d. We take $c' = \max\left\{c, \frac{2r+1}{r+1}\right\}$. We prove this using forward induction for $d \geq 2r$ and backward induction for $d \leq 2r$. For the base case observe that $h_{2r}(c', r) \geq h_{2r}(c, r) \geq 0$. Now assume that $h_d(c, r) \geq 0$ for some $d \geq 2r$. Then

$$h_{d+1}(c', r) = c'^{d+2} - c'^{d+1} - \binom{d+1}{r} = c'(c'^{d+1} - c'^d) - \binom{d+1}{r} \geq c'\binom{d}{r} - \binom{d+1}{r} \geq 0.$$

The second last inequality follows from induction hypothesis while the last inequality follows as: $c' \geq \frac{\binom{d+1}{r}}{\binom{d}{r}} = \frac{d+1}{d+1-r} \geq \frac{2r+1}{r+1}$, for $d \geq 2r$. Similarly using backward induction we can show that $h_d(c', r) \geq 0$ for $d \leq 2r$. Observe that for $r \geq 0$, $1 \leq \frac{2r+1}{r+1} < 2$, is a constant depending on r alone. So now we are left with showing a $c = f(r)$ for $h_{2r}(x, r)$. For $r \geq 5$, we know that $\binom{2r}{r} \leq \frac{2^{2r}}{4}$. We choose a c such that $c^{2r+1} - c^{2r} \geq \frac{2^{2r}}{4}$ which will prove the desired result. We take $c = 2^{1-\frac{1}{2r}}$ for $r \geq 5$ and $c = 1.761$ for $r \leq 4$. For small values of r we get the desired number by directly solving the corresponding equations.

Hence for any $r \geq 0$, we choose $c = \max\left\{1.761,\ 2^{1-\frac{1}{2r}},\ \frac{2r+1}{r+1}\right\}$. This proves that our generic algorithm **Max-r-RIS** takes $\mathcal{O}(c^n)$ time, $c < 2$, for any positive constant r. □

We gave a conservative bound on the value of c in the Theorem 1, as our main aim there was to obtain a $c < 2$ for any fixed constant r. For smaller values of r, we obtain improved bounds on c by directly finding the roots of the polynomials coming from the recurrences of MAX-r-RIS algorithm. Without going into the details, we list c for various values of r in the table below where $\mathcal{O}(c^n)$ is the runtime of our **Max-r-RIS** algorithm.

Table 1. Improved Upper Bounds on c for Various r

$r =$	1	2	3	4	5	6	7	8	9
$c =$	1.69562	1.70688	1.73615	1.76357	1.78554	1.80351	1.81846	1.83111	1.84195

$r =$	10	15	20	30	50	75	100	125	150
$c =$	1.85136	1.88452	1.90486	1.92868	1.95138	1.96458	1.97186	1.97652	1.97979

We observe that the **Max-r-RIS** algorithm can be generalized to solve the counting versions of M-r-RIS problems. The counting version of M-r-RIS problems (#M-r-RIS) asks for the number of maximum r regular induced subgraphs of the given graph

G. We can also consider counting the number of maximal r-regular induced subgraphs of the given graph G which we call #MAXIMAL-r-RIS problems. To solve these problems we allow our algorithm **Max-r-RIS** to enumerate all the R's it finds during the recursion for G and check whether they are maximal if we want to count maximal r-regular induced subgraphs alone. If we want to count maximum r-regular induced subgraphs then we also need to check the size of R. Thus we give the following theorem.

Theorem 2. *Let $G = (V, E)$ be a graph on n vertices and r be a fixed constant. Then (a) #M-r-RIS problems and (b) #MAXIMAL-r-RIS problems can be solved in $\mathcal{O}(c^n)$ time, where c is max of* $\{1.761, 2^{1-\frac{1}{2r}}, (2r+1)/(r+1)\}$.

We observed above that our algorithm enumerates all maximal r-regular induced subgraphs. Hence Theorem 2 also implies that the number of maximal r-regular induced subgraphs of a graph on n vertices is upper bounded by the time complexity of the algorithm. Let $\mathscr{M}_r(n)$ denote the number of maximal r-regular induced subgraph of graphs on n vertices, then we get following theorem.

Theorem 3. *Let $G = (V, E)$ be a graph on n vertices and r be a fixed constant. Then $\mathscr{M}_r(n)$ is upper bounded by c^n, where c is max of* $\{1.761, 2^{1-\frac{1}{2r}}, (2r+1)/(r+1)\}$, *i.e. $\mathscr{M}_r(n)$ is upper bounded by $o(2^n)$, if r is a fixed constant.*

In the next section we consider the lower bounds on the number of maximal r-regular induced subgraphs on graphs on n vertices and improve the upper bounds coming from Theorem 3 to match the lower bounds for various r.

3 Bounds on Number of Maximal r-Regular Induced Subgraphs

Moon and Moser [12] gave a matching lower and upper bound of $3^{n/3}$ on the number of maximal independent sets on a graph on n vertices. We generalize this result and give matching *algorithmic* lower and upper bounds on $\mathscr{M}_r(n)$ for larger values of r.

3.1 Bounds on $\mathscr{M}_1(n)$ or Number of Maximal Induced Matching

For lower bound assume that $n \equiv (0 \mod 5)$. Consider the graph $G = \bigcup_{i=1}^{\frac{n}{5}} K_5^i$ that is $n/5$ disjoint copies of K_5 (K_n^i represents the complete graph on n vertices). Observe that we need to include one edge from each copy of the K_5 (we can include exactly one edge from each copy) to obtain a maximal induced matching for G. Since a K_5 has 10 edges and for any K_5 we can select any edge, we get $10^{n/5}$ distinct maximal induced matching for G, giving a lower bound of $10^{n/5}$ on $\mathscr{M}_1(n)$. This shows the following theorem.

Theorem 4. *$\mathscr{M}_1(n)$ is at least* $10^{n/5} \approx 1.58489^n$.

For an upper bound proof, we obtain recurrences for $\mathscr{M}_1(n)$ by considering various cases based on the maximum degree of the graph. The proof is long and is similar to the upper bound proof in Theorem 6 which we prove in detail below.

Theorem 5. *$\mathscr{M}_1(n)$ is at most $10^{n/5} \approx 1.58489^n$ and all the maximal induced matching of a graph G can be enumerated with polynomial delay.*

3.2 Bounds on $\mathscr{M}_r(n)$ for $r \geq 2$

Now we extend the matching upper and lower bounds for larger values of $r(\geq 2)$. To give the upper bound on $\mathscr{M}_r(n)$, we define the following generalized problem.

> GEN-r-RIS (G-r-RIS): Given a graph $G = (V, E)$ and $R \subseteq V$, such that $G[R]$ is connected induced subgraph of degree at most r. The objective is to find a maximum $R' \subseteq V - R$ such that $G[R \cup R']$ is a r regular subgraph extending R.

Observe that given any instance (G, R), where R satisfies the constraints in the definition of G-r-RIS problem, if we can give a bound on the number of R' such that $G[R' \cup R]$ is a maximal r-regular subgraph then by setting $R = \emptyset$ we have an upper bound on $\mathscr{M}_r(n)$. Given an instance (G, R) where R satisfies the constraints in G-r-RIS problem, we define μ as follows:

$$\mu = \alpha|N^R| + \beta|U|$$

Here $N^R = N[R] - R$ and $U = V - N[R]$. In other words, we assign a weight of α to the vertices of N^R and β to the vertices of U. The value of α and β depend on the problem. The weight of a vertex changes in following situation:

1. If a vertex goes to N^R from U then the weight changes from β to α and the μ changes by $\delta = \beta - \alpha$.
2. If a vertex has current weight either α or β and the vertex is either included in R or removed from the graph then the weight changes to 0. In this case μ changes either by α or β.

We use μ as a measure rather than the number of vertices and give an upper bound on $\mathscr{M}_r(n)$ as a function f of μ. We exemplify the approach by giving the matching lower and upper bound on the number of *maximal* 2*-regular induced subgraphs*.

Theorem 6. *$\mathscr{M}_2(n)$ is at most $35^{n/7} \approx 1.66181^n$ and there exists a graph on n vertices such that $\mathscr{M}_2(n)$ is at least $35^{n/7} \approx 1.66181^n$. Moreover, all the maximal 2-regular induced subgraphs of a graph G can be enumerated with polynomial delay.*

Proof. For the lower bound on $\mathscr{M}_2(n)$, assume that $n \equiv (0 \mod 7)$ and consider the graph $G = \bigcup_{i=1}^{\frac{n}{7}} K_7^i$, $n/7$ disjoint copies of K_7. Any maximal 2-regular induced subgraph of G contains a 2 regular induced subgraph (a triangle) from each copy of K_7. Every K_7 has 35 distinct triangles and hence G has $35^{n/7}$ distinct maximal 2-regular induced subgraphs. This shows the desired lower bound on $\mathscr{M}_2(n)$.

For upper bound, we consider the generalized problem where we have been given $(G = (V, E), R)$ and R satisfies the constraints in the definition of the G-2-RIS problem. We give a bound on the number of R''s, i.e. is the size of the set $\{R' \mid G[R' \cup R]$ is a maximal 2-regular $\}$ as a function f of μ. Depending on various cases we give recurrence relation for f.

Case 1: $(G[R] \neq \emptyset)$ Here we have two cases based on the degree of a vertex in $G[R]$. For a subset $X \subseteq V$, by $deg_X(v)$ we mean the number of neighbors of v in $G[X]$.

Suppose we have a vertex $v \in R$ such that $deg_R(v) = 2$ and have l neighbors in $V - R$ then

$$f(\mu) \leq f(\mu - \alpha l);$$

as none of the l neighbors of v in $V - R$ can be selected in any R' extending R and hence can be removed from the graph, leading to decrease in μ by at least αl. Now suppose we have a vertex v such that degree of v is d in G and $deg_R(v) = 1$.

Now any maximal 2-regular induced subgraph extending R must contain one of the neighbors of v in $V - R$. Hence when we include a neighbor u of v in R we remove all other neighbors of v from G as they can not be part of any R' extending R. This reduces μ by $\alpha(d-1)$. Since there are $d-1$ neighbors of v in $V - R$, we get the following recurrence:

$$f(\mu) \leq (d-1)f(\mu - \alpha(d-1)).$$

We can assume that $(d-1) \geq 1$, otherwise in this case R can not be extended to any maximal 2 regular induced subgraphs resulting in $f(\mu) = 0$.

Case 2: ($R = \emptyset$) We assume that the minimum degree of G is at least 2, as the vertices of degree at most 1 can never be part of any maximal 2 regular induced subgraphs. Also note that every vertex has weight β now. Let v be a vertex of maximum degree d. A maximal 2-regular induced subgraph of G either does not contain v or contains v and its two neighbors. In the first case μ reduces by β and in the other cases where v and its two neighbors are selected in R and other neighbors of v are removed from the graph, μ decreases by $(d+1)\beta$. This gives the following worst case recurrence on $f(\mu)$:

$$f(\mu) \leq f(\mu - \beta) + \binom{d}{2} f(\mu - (d+1)\beta).$$

When $d \geq 7$ this recurrence itself gives us the desired bound on $\mathscr{M}_2(n)$. So from now on we assume that the maximum degree of G is at most 6. To obtain the desired bound in this case we refine the recurrences on $f(\mu)$ based on following three cases. These cases are applied in order of their appearance.

(a) CON-COM CASE: There exists a vertex v such that $G[N[v]]$ is one of the connected component of G. Call the connected component containing v $\mathcal{C}_v$. Now the number of maximal 2-regular induced subgraphs of G is maximized when we have $\mathcal{C}_v$ such that $\mathcal{C}_v$ has maximum number of maximal 2-regular induced subgraphs. This happens precisely when $\mathcal{C}_v = K_t$ where $t = deg_V(v)$. So for this case we get:

$$f(\mu) \leq \binom{d+1}{3} f(\mu - \beta(d+1)), \ \ 2 \leq d \leq 6.$$

(b) CUT-EDGE CASE: We have a vertex v such that it has an unique neighbor u having an unique neighbor x such that $x \notin N[v]$. Since the edge (u, x) is a cut edge it is not part of any maximal 2-regular induced subgraph. So the number of maximal 2 regular subgraphs of G is upper bounded by the number of maximal 2 regular subgraphs of G' obtained from G by removing the edge (u, x). This reduces it to CON-COM CASE.

(c)AT-LEAST-2-IN-$N_2[v]$ CASE: In this case every vertex $v \in V$ either has a neighbor u such that u has at least 2 neighbors not in $N[v]$ or there are at least two neighbors of v which don't have neighbors in $N[v]$. For this case we give a generic recurrence. Partition the neighbor set $N(v)$ of v into W_1, W_2 and W_3 such that every vertex $u \in W_1$ has $N(u) \subseteq N[v]$, each vertex in W_2 has an unique neighbor x such that $x \notin N[v]$ while every vertex $u \in W_3$ has at least 2 neighbors not in $N[v]$. By S^v_y we mean the set $N(y)$-$N[v]$. Let $2 \leq \sum_{i=1}^{3} |W_i| = d \leq 6$. We consider the recurrence on $f(\mu)$ based on whether or not v is a part of maximal 2-regular induced subgraph. When $v \notin R$ μ changes by $\mu-\beta$. Now we consider the case when v and its two neighbors u_1, u_2 and $u_1 \neq u_2$ are in R and see the change in μ based on which W_i's, $1 \leq i \leq 3$, u_1 and u_2 belong.

- **(A)** $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_1} \times \mathbf{W_1}]$ μ changes to $\mu - \beta(d+1)$.
- **(B)** $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_1} \times \mathbf{W_2}]$ The only way we can have a 2-regular induced subgraph is when (u_1, u_2) is an edge and v, u_1, u_2 is a triangle. This implies that x, the unique neighbor of u_2 not in $N[v]$ will be removed from the graph. This reduces μ to $\mu - \beta(d+1) - \beta$.
- **(C)** $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_1} \times \mathbf{W_3}]$ Similar to the previous case we can argue that μ at least reduces to $\mu - \beta(d+1) - 2\beta$.
- **(D)** $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_2} \times \mathbf{W_2}]$ The worst case is when u_1 and u_2 have a common neighbor x which is not in $N[v]$. In this case μ changes to $\mu - \beta(d+1) - \beta$.
- **(E)** $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_2} \times \mathbf{W_3}]$ If (u_1, u_2) is an edge or u_1 and u_2 have a common neighbor x then either $\{v, u_1, u_2\}$ or $\{v, u_1, u_2, x\}$ forms a 2 regular induced subgraph leading to a reduction of $\beta(d+1) - 2\beta$ in μ. When none of these cases arise then since x is an unique neighbor of u_1, x gets included in R and two neighbor of u_2 become elements of N^R, leading to change in μ by $\beta(d+1) - \beta - 2\delta$.
- **(F)** $[(\mathbf{u_1}, \mathbf{u_2}) \in \mathbf{W_3} \times \mathbf{W_3}]$ Here the worst case is when u_1 and u_2 have exactly two neighbors not in $N[v]$ and $S^v_{u_1} = S^v_{u_2}$, that is u_1 and u_2 have common neighbors not in $N[v]$. This reduces μ by $\beta(d+1) - 2\delta$ as both neighbors of u_1 and u_2 which are not in $N[v]$ become element of N^R.

Above discussion gives us following recurrence on $f(\mu)$.

$$\begin{aligned} f(\mu) \leq f(\mu-\beta) &+ \binom{|W_1|}{2} f(\mu - \beta(d+1)) + |W_1||W_2| f(\mu - \beta(d+1) - \beta) \\ &+ |W_1||W_3| f(\mu - \beta(d+1) - 2\beta) + \binom{|W_2|}{2} f(\mu - \beta(d+1) - \beta) \\ &+ |W_2||W_3| f(\mu - \beta(d+1) - \beta - 2\delta) + \binom{|W_3|}{2} f(\mu - \beta(d+1) - 2\delta). \end{aligned}$$

We assume that $\binom{l_1}{l_2} = 0$ if $l_1 < l_2$. Note that, $|W_1| \leq d-1$ and if there is an unique neighbor u of v having a neighbor x such that $x \notin N[v]$ then $W_2 = \emptyset$ because of the CUT-EDGE CASE.

We numerically obtain $\alpha = 1.45$, $\beta = 2$ and $\delta = \beta - \alpha = 0.55$, as values which minimizes the above set of recurrences on f.

We used a program to generate the above set of recurrences based on different partitions of $N(u)$ and found that the worst case recurrence among the above set after setting $\alpha = 1.45$ and $\beta = 2$ corresponds to the following scenario:

$$d = 5,\ W_1 = W_2 = \emptyset \text{ and } \forall (y,z) \in W_3 \times W_3,\ |S^v_y \cup S^v_z| = 2.$$

The recurrence corresponding to this scenario is: $f(\mu) \leq f(\mu - \beta) + 10 f(\mu - 6\beta - 2\delta)$.

r	3	4	5	6	7	8	10	15
lb_r	1.71149	1.7468	1.7734	1.7943	1.8113	1.8253	1.8474	1.8828
ub_r	1.73615	1.76357	1.78554	1.80351	1.81846	1.83111	1.85136	1.88452
$ub_r - lb_r$	0.02466	0.016782	0.012131	0.0091762	0.0071727	0.0057618	0.0039415	0.0017377

Fig. 2. Bounds on the Number of Maximal-r-Regular Induced Subgraphs for Small Values of r

All the recurrences occurring in all the above cases (Cases 1 & 2) are dominated by

$$f(\mu) \leq \binom{7}{3} f(\mu - 7\beta)$$

which solves to $(35)^{\frac{\mu}{7\beta}}$. Now given a graph G, $\mu(G) \leq n\beta$, and hence

$$\mathscr{M}_2(n) \leq f(\beta n) \leq 35^{\beta n/7\beta} = 35^{n/7}.$$

This proves the required upper bound. These cases can be changed in branching steps leading to an enumeration algorithm running in $\mathcal{O}(35^{n/7}) = \mathcal{O}(1.66181^n)$ time. □

To obtain a lower bound on $\mathscr{M}_r(n)$ for larger values of r we need to find a function $g(r)$ such that when we take G as $\frac{n}{g(r)}$ disjoint copies of $K_{g(r)}$ then $\binom{g(r)}{r+1}^{1/g(r)}$ is maximized. We obtain the following description for $g(r)$.

Lemma 2. *Given r, $g(r)$ defined below*

$$g(r) = \begin{cases} 2r+3 & 0 \leq r \leq 11 \\ 2r+4 & 12 \leq r \leq 100 \\ 2r+2+\left\lfloor \frac{1}{2}\ln\left(\frac{(2r+1)\pi}{2}\right) + O\left(\frac{(\ln r)^2}{r}\right)\right\rfloor & r > 100 \end{cases}$$

maximizes $\binom{g(r)}{r+1}^{1/g(r)}$. Hence $\mathscr{M}_r(n)$ is at least $\binom{g(r)}{r+1}^{n/g(r)}$.

The proof of Lemma 2 is based on estimates on binomial coefficients and will appear in the longer version of the paper.

For a fixed r, let lb_r and ub_r denote a base of exponent in lower bound and upper bound on $\mathscr{M}_r(n)$, i.e., $lb_r^n \leq \mathscr{M}_r(n) \leq ub_r^n$. When $r \geq 3$, we obtain tighter upper bounds on $\mathscr{M}_r$ by directly finding the roots of the polynomials coming from the recurrences in MAX-r-RIS algorithm. We can see that the upper bound obtained this way and the lower bound coming from Lemma 2 are already very close, as Figure 2 shows. For small values of r, these upper bounds could be made equal to lower bound by choosing α and β appropriately in the definition of μ and by doing the analysis similar to the one in Theorem 6. For an example, when $r = 3$ we can take $\alpha = 1.73$ and $\beta = 2$ and show that $lb_3 = ub_3$. We do not go into the details due to lack of space and the details will appear in the full version of the paper.

4 Improved Algorithms for $r = 1$ and 2 and Applications

Our generic algorithm **Max-r-RIS** finds a maximum r-regular induced subgraph in time $\mathcal{O}(1.6957^n)$ and $\mathcal{O}(1.7069^n)$ for $r = 1$ and 2 respectively. Our algorithmic upper bound proofs (on $\mathscr{M}_r(n)$) of Section 3 enumerates all maximal r-regular subgraphs in time $\mathcal{O}(1.58469^n)$ and $\mathcal{O}(1.66181^n)$ for $r = 1$ and 2 respectively, already improving the bounds given in Section 2. Here we further improve these algorithms for $r = 1$ and 2 and give an application of algorithms developed so far in the paper.

4.1 Maximum Induced Matching (MIM) and M-2-RIS Problems

Let $G = (V, E)$ be a graph and v be a vertex having a neighbor u such that $N(u) \subseteq N[v]$. Consider the set $\mathcal{M}_v$ of maximum sized induced matching having v (these may not be the maximum sized induced matching of G). Then the following is easy to see.

Lemma 3. *Let G be a graph and v be a vertex and $u \in N(v)$ such that $N(u) \subseteq N[v]$. Then there exists a $M' \in \mathcal{M}_v$ such that it contains the edge (v, u).*

The other observation relates MIM of G to MAXIMUM INDEPENDENT SET (MIS) of square of the *line graph* of G. The *line graph*, $L(G)$ of $G = (V, E)$ is the graph whose vertices are edges of G, and two edges e_1, e_2 are adjacent if and only if they are adjacent edges in G. G^i (ith power of G) is a graph on V and there are edges between two vertices v_1 and v_2 if and only if there is a path of length at most i between v_1 and v_2.

Lemma 4 ([4]). *Let G be a graph then $MIM(G) = MIS(L(G)^2)$.*

So our algorithm uses branching on a vertex v when the maximum degree of the graph is at least 5 and distinguishes cases based on Lemma 3. When the maximum degree of the graph is at most 4, we use the well known algorithms to find a maximum independent set [7,15] in $L(G)^2$. Without going into further details we state the following theorem.

Theorem 7. *Let $G = (V, E)$ be a graph on n vertices, then a* MIM *can be found in (a) $\mathcal{O}(1.4904^n)$ time and space polynomial in n or in (b) $\mathcal{O}(1.4786^n)$ time and space exponential in n.*

We obtain an improved algorithm for M-2-RIS by refining the measure defined in the Section 3 and by using new branching rules. We omit the details and simply state the following theorem.

Theorem 8. *Let $G = (V, E)$ be a graph on n vertices, then the* MAXIMUM 2-REGULAR INDUCED SUBGRAPH *problem can be solved in $\mathcal{O}(1.62355^n)$ time.*

4.2 Induced r-Regular Subgraph Isomorphism

Here we consider a special case of INDUCED SUBGRAPH ISOMORPHISM (IND-SI) problem.

> IND-SI: Given a graph $G = (V, E)$ and H, the question is to determine whether there exists a $H' \subseteq V$ such that $G[H'] \cong H$.

A brute force algorithm for this is to enumerate all subsets H' of size $|H|$ of G and check whether $G[H'] \cong H$, using the $O(n^{o(n)})$ time graph isomorphism algorithm [1]. The question is: *can we do this in time $\mathcal{O}(c^n)$ time where n is the number of vertices* *in G and $c < 2$, a constant?* Here, we answer this question for a special class of H, that is when *H is a r-regular graph with r a constant*. Even with such restrictions this problem is NP-hard as it contains problems like INDEPENDENT SET. We show the following theorem.

Theorem 9. *Given a graph $G = (V, E)$ on n vertices and a graph H, where H is r-regular, for a constant r, we can determine whether there exists a $H' \subseteq V$ such that $G[H'] \cong H$ in $\mathcal{O}(c^n)$ time, where $c < 2$ a constant depending on r alone.*

Proof. Let $H = \{H_1, H_2, \cdots, H_r\}$ where each H_i is a connected component of H. If there exists a $H' \subseteq V$ such that $G[H'] \cong H$ then H' can also be written as $\{H'_1, H'_2, \cdots H'_r\}$, H'_i connected component of $G[H']$, such that $G[H'_i] \cong H_i$ for $1 \le i \le r$.

The crucial observation is that if there exists a H' such that $G[H'] \cong H$ then there exists a maximal r-regular induced subgraph R extending H' such that each of the connected component of H' appears as a connected component of $G[R]$. By applying Theorem 3, we enumerate all maximal r-regular induced subgraphs of a graph on n vertices in $\mathcal{O}(c^n)$ time, $c < 2$ a constant depending on r alone. Now given a R, a maximal r-regular induced subgraph of G, we check the isomorphism of each connected component of $G[R]$ with each of H_i using the polynomial time bounded degree graph isomorphism algorithm of Luks [11]. If we obtain a H' such that $G[H'] \cong H$ then we return H' else we return no. The correctness and the time complexity of the algorithm follow easily. □

5 Conclusion

In this paper we developed an $\mathcal{O}(c^n)$ time exact algorithms for MAXIMUM r-REGULAR INDUCED SUBGRAPH problems for any fixed constant r, where $c < 2$ is a constant depending on r alone. We also showed that if r is a constant then the number of maximal r-regular induced subgraphs on a graph on n vertices is bounded by $o(2^n)$. Then we gave very tight lower and upper bounds on the number of maximal r-regular induced subgraphs on n vertices. All our algorithms were simple to describe but their analyses were non-trivial and involved a different measure than the usual number of vertices to measure the progress of the algorithms. We analyzed recurrences having binomial coefficients and believe that these may trigger some new results in the area of exact algorithms. Finally, we used the results obtained on the enumeration version of MAXIMUM r-REGULAR INDUCED SUBGRAPH problems to give a non trivial exact algorithm for INDUCED r-REGULAR SUBGRAPH ISOMORPHISM when r is a constant. The other problems for which we can give non trivial exact algorithms based on the algorithms and the techniques developed in this paper include EFFICIENT EDGE DOMINATING SET [10], δ-SEPARATING MAXIMUM MATCHING [17] and MAXIMUM BOUNDED DEGREE INDUCED SUBGRAPH problems.

It will be interesting to find other applications of the algorithms developed in this paper. Finding a non trivial exact algorithm for INDUCED SUBGRAPH ISOMORPHISM problem, even for special classes of H, remains open. Here we obtained an efficient algorithm for INDUCED SUBGRAPH ISOMORPHISM when H is a r-regular graph for a constant r.

Acknowledgments. The last author thanks R. Balasubramanian for several discussions on various parts of the paper.

References

1. L. BABAI, W. M. KANTOR AND E. M. LUKS. *Computational Complexity and the Classification of Finite Simple Groups.* In the Proceedings of FOCS'83. 162-171 (1983).
2. V. BONIFACI, U. D. IORIO AND L. LAURA. *On the Complexity of Uniformly Mixed Nash Equilibria and Related Regular Subgraph Problems.* In the Proceedings of FCT'05. LNCS 3623: 197-208 (2005).
3. J. M. BYSKOV. *Enumerating Maximal Independent Sets with Applications to Graph Colouring.* Operations Research Letters 32(6): 547-556 (2004).
4. K. CAMERON. *Induced Matchings.* Discrete Applied Mathematics 24: 97-102 (1989).
5. D. M. CARDOSO, M. KAMINSKI AND V. LOZIN. *Maximum k-Regular Induced Subgraphs.* Rutcor Research Report (RRR) 3, (2006).
6. F. V. FOMIN, S. GASPERS, AND A. V. PYATKIN. *Finding a Minimum Feedback Vertex Set in time $O(1.7548^n)$.* In the Proceedings of IWPEC'06. LNCS 4169: 184-191 (2006).
7. F. V. FOMIN, F. GRANDONI, AND D. KRATSCH. *Measure and Conquer: A Simple $O(2^{0.288n})$ Independent Set Algorithm.* In the Proceedings of SODA'06: 18-25(2006).
8. F. V. FOMIN, F. GRANDONI, AND D. KRATSCH. *Some new techniques in design and analysis of exact (exponential) algorithms.* Bulletin of the EATCS 87: 47-77 (2005).
9. M. R. GAREY AND D. S. JOHNSON. *Computer and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA., (1979).
10. J. P. GEORGES, M. D.HALSEY, A. M. SANAULLA, M. A. WHITTLESEY. *Edge Domination and Graph Structure.* Cong. Numer. 76: 127-144 (1990).
11. E. M. LUKS. *Isomorphism of Graphs of Bounded Valence can be Tested in Polynomial Time.* Journal of Computer System Sciences 25(1): 42-65 (1982).
12. J. W. MOON AND L. MOSER. *On Cliques in Graphs.* Israel Journal of Mathematics 3: 23-28 (1965).
13. V. RAMAN, S. SAURABH AND S. SIKDAR *Efficient Exact Algorithms through Enumerating Maximal Independent Sets and Other Techniques.* To appear in Theory of Computing Systems.
14. I. RAZGON. *Exact Computation of Maximum Induced Forest.* In the Proceedings of SWAT'06. LNCS 4059: 160-171 (2006).
15. J. M. ROBSON. *Algorithms for Maximum Independent Set.* Journal of Algorithms 7: 425 - 440 (1986).
16. A. STEGER AND M. YU. *On Induced Matchings.* Discrete Mathematics 120: 291-295 (1993).
17. L. J. STOCKMEYER AND V. V. VAZIRANI. *NP-Completeness of Some Generalizations of the Maximum Matching Problem.* Information Processing Letters 15(1): 14-19 (1982).
18. G. WOEGINGER. *Exact algorithms for NP-hard problems: A survey.* In *Combinatorial Optimization—Eureka! You shrink!.* LNCS 2570: 185-207 (2003).

Solving Connected Dominating Set Faster Than 2^n

Fedor V. Fomin[1], Fabrizio Grandoni[2], and Dieter Kratsch[3]

[1] Department of Informatics, University of Bergen, N-5020 Bergen, Norway
fomin@ii.uib.no

[2] Dipartimento di Informatica, Università di Roma "La Sapienza",
Via Salaria 113, 00198 Roma, Italy
grandoni@di.uniroma1.it

[3] LITA, Université Paul Verlaine - Metz, 57045 Metz Cedex 01, France
kratsch@univ-metz.fr

Abstract. In the connected dominating set problem we are given an n-node undirected graph, and we are asked to find a minimum cardinality connected subset S of nodes such that each node not in S is adjacent to some node in S. This problem is also equivalent to finding a spanning tree with maximum number of leaves.

Despite its relevance in applications, the best known exact algorithm for the problem is the trivial $\Omega(2^n)$ algorithm which enumerates all the subsets of nodes. This is not the case for the general (unconnected) version of the problem, for which much faster algorithms are available. Such difference is not surprising, since connectivity is a global property, and non-local problems are typically much harder to solve exactly.

In this paper we break the 2^n barrier, by presenting a simple $O(1.9407^n)$ algorithm for the connected dominating set problem. The algorithm makes use of new domination rules, and its analysis is based on the Measure and Conquer technique.

1 Introduction

Nowadays, it is common belief that NP-hard problems cannot be solved in polynomial time. For a number of NP-hard problems, we even have strong evidence that there are no sub-exponential algorithms [19]. Moreover, many relevant problems do not admit satisfactory approximation algorithms. For example, Maximum Independent Set is hard to approximate within $n^{1-\varepsilon}$ [17]. For these problems a promising approach is to design exact algorithms with smallest possible exponential running times.

The recent interest in exact exponential algorithms has several motivations. Indeed, there are applications that require exact solutions of NP-hard problems, although this might only be possible for moderate input sizes. Decreasing the exponential running time, say, from $\mathrm{O}(2^n)$ to $\mathrm{O}(2^{0.9\,n})$, increases the size of the instances solvable within a given amount of time by a constant *multiplicative* factor. This kind of improvements can be crucial in several applications. On the other hand, the study of exact algorithms leads to a better understanding of NP-hard problems, and initiates new combinatorial and algorithmic challenges.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 152–163, 2006.

In this paper we consider one of the classical NP-hard problems, the Connected Dominating Set problem (CDS). A *connected dominating set* of a graph $G = (V, E)$ is a subset of nodes $S \subseteq V$ such that S is a dominating set of G and the subgraph of G induced by S is connected. The Connected Dominating Set problem asks to find a connected dominating set of smallest possible cardinality. This problem is also equivalent to finding a spanning tree with maximum number of leaves. Connected Dominating Set is a fundamental problem in connected facility location and studied intensively in computer science and operations research [16,26]. Another recent application of this problem is in wireless ad-hoc networks: a small connected dominating set is often a convenient backbone to route the flow throughout the network (see e.g. [2]). The problem is NP-hard [13] and there is a $(\ln \Delta + O(1))$-approximation algorithm, where Δ is the maximum degree [15]. Such approximation guarantee cannot be improved unless $NP \subseteq DTIME(n^{O(\log \log n)})$ [15]. Despite its relevance in applications, the current best exact algorithm for Connected Dominating Set is the trivial $\Omega(2^n)$ enumerative algorithm, which tries all possible subsets of nodes. Better results are known for the general (unconnected) version of the problem [8,12,14,22]: the current best algorithm for Dominating Set has running time $O(2^{0.598n})$ [8].

Though apparently closely related, Connected Dominating Set and Dominating Set are rather different from the point of view of exact algorithms. In particular, the techniques used to solve Dominating Set do not seem to work for Connected Dominating Set. One of the main reasons of this discrepancy is that *connectivity* is a global property: very often exact algorithms are based on the local structure of the problem; these algorithms seem not able to capture global properties such as connectivity.

Indeed, Connected Dominating Set belongs to a family of *non-local* problems which turns out to be particularly hard to solve exactly. Probably the best known example of this kind of problems is the Travelling Salesman Problem: find a minimum cost tour which visits all the nodes of a weighted graph. The fastest known algorithm for this problem, which dates back to the sixties [18], is based on dynamic programming and has running time $\Omega(2^n)$. Better results are known only for special graph classes, such as cubic graphs [6]. For many other non-local problems the current best known algorithms are still trivial. There are only a few exceptions to this. A relevant example is Steiner Tree: find a minimum size subtree of a given graph spanning a given subset of k nodes. For this problem an $O(1.4143^n)$ time algorithm can be obtained by combining the $O((2+\epsilon)^k n^{O(1)})$ dynamic-programming (exponential space) algorithm in [21] (for small k), with trivial $O(2^{n-k} n^{O(1)})$ enumeration of Steiner nodes (for large k). Finding a polynomial space algorithm faster than 2^n is still open. Another very recent example is a $O(1.9053^n)$ algorithm for Feedback Vertex Set: find a minimum cardinality subset of nodes of a graph whose removal makes the graph acyclic [23].

Our results. In this paper we make a further significant step in the design of faster exact algorithms for non-local problems, by presenting the first algorithm for Connected Dominating Set which breaks the 2^n barrier: our recursive algorithm takes polynomial space and runs in time $O(1.9407^n)$. The algorithm is based on the simple strategy "stay connected", which means that all partial solutions generated recursively must be connected. Local choices are performed under this constraint. Our algorithm makes use of new domination rules, which were designed with the stay-connected framework in mind.

If analyzed in the standard way, our algorithm performs very poorly. The refined time bound is obtained with the Measure and Conquer approach described in [8]. The idea is to lower bound the progress made by the algorithm at each branching step according to non-standard measures of the size of the subproblems. However, the measure used in [8] for Dominating Set does not seem to work properly here. For this reason we designed a new, non-trivial measure: for every vertex v our measure reflects both the "need for domination" of v, and the ability of v "to dominate" the vertices that are not dominated yet. We remark that here Measure and Conquer is crucial to break the 2^n barrier. Moreover, we believe this approach is flexible enough to be applied to other non-local problems.

Measure and Conquer does not necessarily provide tight upper bounds for the worst case running time of recursive exponential algorithms, thus lower bounds are of great interest. As a second contribution of this paper, we establish a lower bound of $\Omega(4^{n/5})$ for the worst case running time of our algorithm.

Related Work. The design of exponential time algorithms has a long history dating back to Held and Karp's paper [18] on the travelling salesman problem in the early sixties. The last years have seen an emerging interest in constructing exponential time algorithms for combinatorial problems like Coloring [1,4], Max-Cut [27], 3-SAT [3,5], Dominating Set [8], Treewidth [11], and Independent Set [10]. There are two nice surveys of Woeginger [28,29] describing the main techniques that have been established in the field. We also recommend the survey of Iwama [20] devoted to exponential algorithms for 3-SAT and the paper of Schöning [25] for its introduction to exponential time algorithms. In [9] we review some new techniques for the design and analysis of exponential-time algorithms, among which "Measure and Conquer" and "Lower Bounds".

One of the major techniques for constructing fast exponential time algorithms, which is also used in our CDS algorithm, is the *Branch and Reduce* paradigm. Roughly speaking, Branch and Reduce algorithms (also called search tree algorithms, Davis-Putnam-style exponential-time backtracking algorithms etc.) first apply some reduction rules, and then branch on two or more subproblems, which are solved recursively. Their running time analysis is based on a measure for the problem instance; reduction and branching rules lead to linear recurrences in the measure and their solution by standard methods provides upper bounds for the worst case running time. Recently, non-standard measures for problem instances have been used to improve the analysis of Branch and Reduce algorithms. This approach is called Measure and Conquer in [8]. The analysis of our algorithm for CDS is heavily based on this technique.

2 The Algorithm

Let $G = (V, E)$ be an n-node undirected and simple graph. The open *neighborhood* of a node v is denoted by $N(v) = \{u \in V : uv \in E\}$, and the closed neighborhood of v is denoted by $N[v] = N(v) \cup \{v\}$. The subgraph of G induced by a set $S \subseteq V$ is denoted by $G[S]$. A set $S \subseteq V$ of nodes of G is *connected*, if $G[S]$ is connected.

Without loss of generality, we can assume (i) that the graph is connected (otherwise there is no solution) and (ii) the minimum connected dominating set has cardinality at least two (otherwise the problem is trivially solvable in polynomial time). By the last assumption, we can consider the *total* variant of CDS, where each node v *dominates* its neighbors $N(v)$, but not the node v itself. This will turn out to be useful in the analysis.

Our recursive CDS algorithm is based on the following approach. Suppose we are given two subsets of nodes S (*selected* nodes), and D (*discarded* nodes), where $|S| \geq 2$ and $G[S]$ is connected. We will describe a recursive algorithm which finds an optimum solution OPT, if any, under the constraint that all the nodes in S and no node in D belong to OPT:

$$S \subseteq OPT \quad \text{and} \quad D \cap OPT = \emptyset.$$

In order to solve CDS it is sufficient to guess two adjacent nodes v' and v'' of some optimum solution, and run the algorithm above on the instance $(S, D) = (\{v', v''\}, \emptyset)$. So we run the algorithm $O(n^2)$ times.

Clearly, the instance is infeasible when $V \setminus D$ is not a connected dominating set. For notational convenience, we will sometimes allow S and D to overlap, and in that case we say that the instance is infeasible as well.

Before describing the algorithm, let us introduce some notation. The *available* nodes $A = V \setminus (S \cup D)$ are the nodes which are neither selected nor discarded. An available node v is a *candidate* if it is adjacent to S, and a *promise* if its removal makes the instance infeasible, i.e. $V \setminus (D \cup \{v\})$ is not a connected dominating set of G. Intuitively, a candidate is a node that might be added to S in the current step, while a promise is a node that must be added to S at some point (if the instance is feasible). We say that a node is *dominated* if it is adjacent to some node in S, and *free* otherwise. By F we denote the set of the free nodes

$$F = V \setminus \cup_{v \in S} N(v).$$

The algorithm halts if either the instance is infeasible or S is a (connected) dominating set. In the first case the algorithm returns *no*, while in the second one it returns $OPT = S$. Otherwise the algorithm performs some reductions on the problem instance, and then it branches on one or more subproblems, which are solved recursively. In each subproblem the algorithm adds available nodes to either S or D but always keeping S connected. The best solution of the subproblems, that is the one which minimizes the size $|OPT|$ of the solution returned, is the solution to the original problem.

The reduction rules are:

(a) If there is a candidate v which is a promise, select it (add it to S);

(b) If there are two candidates v and w (which by (a) are not promises) such that $N(v) \cap F \subseteq N(u) \cap F$, discard v (add it to D);

(c) If there is an available node v which does not dominate any free node, discard v.

The algorithm branches according to the following rules:

(A) If there is a candidate v which dominates at least three free nodes w_1,w_2 and w_3, or which dominates an available node w such that, after selecting v, w does not dominate any free node, branch on the two subproblems

- $(S_1, D_1) = (S \cup \{v\}, D)$;
- $(S_2, D_2) = (S, D \cup \{v\})$.

(B) If there is a candidate v which dominates a unique free node w, let

$$U = \{u_1, u_2, \ldots, u_k\} = N(w) \cap A \setminus N[v]$$

be the set of the available neighbors of w which are not in the closed neighborhood of v. Branch on the three subproblems:

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w\}, D)$;
- $(S_3, D_3) = (S \cup \{v\}, D \cup \{w\} \cup U)$.

Observe that w might be discarded or a promise. Moreover one of the u_i's could be a promise. In those cases one or more subproblems are infeasible, and the algorithm simply halts on such infeasible subproblems. The same kind of situation may happen also in the following cases.

(C) If there is a candidate v which dominates two free nodes w_1 and w_2, name w_1 and w_2 such that if w_2 is available (a promise), so is w_1. Let

$$U_i = \{u_{i,1}, u_{i,2}, \ldots, u_{i,k_i}\} = N(w_i) \cap A \setminus N[v]$$

be the available neighbors of w_i which are not in the closed neighborhood of v. There are three different subcases:

(C.1) If w_1 and w_2 are adjacent, w_1 is available and w_2 is discarded, branch on the three subproblems:

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w_1\}, D)$;
- $(S_3, D_3) = (S \cup \{v\}, D \cup \{w_1\} \cup U_1)$.

(C.2) If w_1 and w_2 are adjacent and both available, branch on the four subproblems:

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w_1\}, D)$;
- $(S_3, D_3) = (S \cup \{v, w_2\}, D \cup \{w_1\})$;
- $(S_4, D_4) = (S \cup \{v\}, D \cup \{w_1, w_2\} \cup U_1 \cup U_2)$.

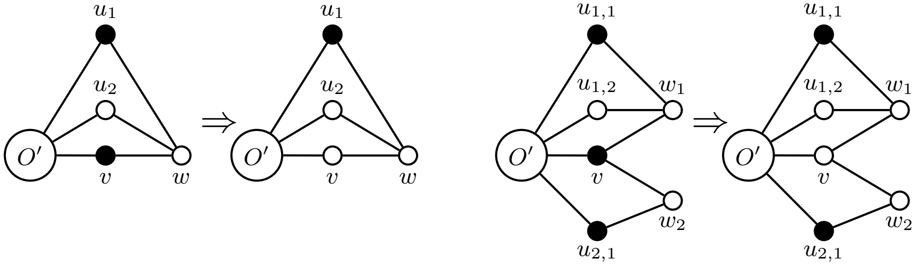


Fig. 1. Examples of cases (B) and (C.3). Black nodes are selected.

(C.3) Otherwise (either w_1 and w_2 are not adjacent, or they are adjacent and both discarded), branch on the five subproblems

- $(S_1, D_1) = (S, D \cup \{v\})$;
- $(S_2, D_2) = (S \cup \{v, w_1\}, D)$;
- $(S_3, D_3) = (S \cup \{v, w_2\}, D \cup \{w_1\})$;
- $(S_4, D_4) = (S \cup \{v\}, D \cup \{w_1, w_2\} \cup U_1)$;
- $(S_5, D_5) = (S \cup \{v\}, D \cup \{w_1, w_2\} \cup U_2)$.

Theorem 1. (correctness) *The algorithm above computes a minimum cardinality connected dominating set.*

Proof. The correctness of the halting rules is trivial.

A reduction rule is *feasible* if it does not modify the value of the optimum. Reduction rule (a) is feasible since removing a candidate v which is a promise would lead to an infeasible instance. Reduction rule (b) is feasible since if $v \in OPT$, then $OPT' = OPT \cup \{w\} \setminus \{v\}$ is a feasible solution of cardinality at most $|OPT|$. Reduction (c) is feasible since all the available neighbors of v are already connected to S, and thus removing v from any feasible solution keeps the solution feasible.

Let us consider the branching rules. First observe that, as required, every set S_i induces a connected subgraph of the original graph. A branching rule is *feasible* if at least one subproblem preserves the value of the optimum solution. Branching rule (A) is trivially feasible: every connected dominating set either contains candidate v or does not. This simple fact is also used in the remaining branching rules.

Consider now branching rule (B). It is sufficient to show that if we select v and discard w, then we must also discard U. Assume by contradiction that $OPT = O' \cup \{v, u_i\}$ is an optimum solution of $(S', D') = (S \cup \{v\}, D \cup \{w\})$, where $u_i \in U$. Since w is discarded, $OPT' = O' \cup \{u_i\}$ is also connected. Moreover, since v dominates only w, and w is dominated by u_i as well, we have that OPT' is a dominating set (see Figure 1). Thus OPT' is a connected dominating set of size $|OPT| - 1$, which is a contradiction.

The feasibility of (C.3) follows by observing that if we select v and discard both w_1 and w_2, then we must also discard either U_1 or U_2 (or both). This can be proved by essentially the same argument as in case (B).

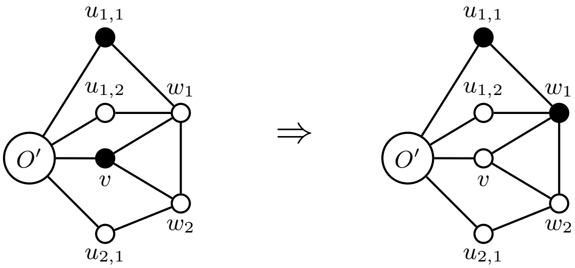


Fig. 2. Example of cases (C.1) and (C.2). Here we are assuming that w_1 is available.

The remaining two cases are slightly more complicated. Consider first case (C.2). It is sufficient to show that, if we select v and discard both the w_i's, then we can also discard U_1 and U_2. By the same argument used in case (C.3), we already know that in the optimum solution OPT to $(S \cup \{v\}, D \cup \{w_1, w_2\})$ we must discard either U_1 or U_2. For sake of contradiction, suppose that $OPT = O' \cup \{v, u_{1,i}\}$ contains one $u_{1,i} \in U_1$ and no node in U_2 (a symmetric analysis holds if OPT contains one $u_{2,j} \in U_2$ and no node in U_1). Since w_1 and w_2 are adjacent, and w_1 is available, we have that by replacing v with w_1 in OPT, we obtain another feasible solution of the same cardinality (see Figure 2). Thus we do not need to consider this case because if OPT is the optimum solution to the original problem, the algorithm will find a solution of the same cardinality while solving subproblem $(S_1, D_1) = (S, D \cup \{v\})$.

Basically the same argument shows that in case (C.1), if we select v and we discard both w_1 and w_2, then we can also discard U_1. Hence the feasibility of (C.1). Note that, differently from case (C.2), we cannot use a symmetric argument to show that also U_2 can be discarded. This is because $w_2 \in D$, and thus the optimum solution to $(S_1, D_1) = (S, D \cup \{v\})$ cannot contain w_2. □

3 Analysis

Consider a given instance (S, D) of the problem (where the graph G is fixed). We will measure the size of the problem as described below. This measure will be used to bound the progress made by the algorithm at each branching step.

We associate two different weights to each node v of the graph. The first weight $\alpha(v) \geq 0$ is used to take into account the need for domination of v. In particular, if v is already dominated by S, $\alpha(v) = 0$. The second weight $\beta(v) \geq 0$ instead reflects the capability of v to dominate free nodes. For this reason, we assume $\beta(v) = 0$ if either $v \in S$ or $v \in D$.

Altogether the weight of the problem is

$$k = k(G, S, D) = k(S, D) = \sum_{v \in F} \alpha(v) + \sum_{v \in A} \beta(v). \qquad (1)$$

In order to simplify the analysis, we make the following assumptions:

- for a given available node v, $\beta(v) = \beta \in (0, 1]$ if v is a promise and $\beta(v) = 1$ otherwise.
- $\alpha(v) = \alpha_{|v|}$ is a non-decreasing function of the *frequency* of v, denoted by $|v|$, that is the number of available nodes which dominate v (recall that v does not dominate itself). More precisely we assume

$$0 = \alpha_0 = \alpha_1 < \alpha_2 < \alpha_3 = \alpha_i, \quad \forall i \geq 4.$$

The reasons for the simplifying assumptions above will be clearer from the analysis. Note that the size of the original problem is upper bounded by $(1 + \alpha_3)n$.

For notational convenience, we define

$$\Delta\alpha_i = \alpha_i - \alpha_{i-1}, \quad i \geq 1.$$

Intuitively, $\Delta\alpha_i$ is the reduction of the size of the problem due to the reduction from i to $i - 1$ of the frequency of a free node.

Fact 1. *Observe that when we discard a candidate node v, we decrease the size of the problem (i) by $\beta(v)$, because v is not available any more, and (ii) by $\Delta\alpha_{|w|}$ for each free neighbor w of v, because of the decrease of the frequency of w. Moreover, the size could further decrease (iii) by $(1 - \beta)$, if some available node becomes a promise.*

On the other hand, when we select a node v, we decrease the size of the problem (i) by $\beta(v)$, because v is not available any more, and (ii) by $\alpha_{|w|}$ for each free neighbor w of v, because w is not free any more.

Fact 1 will be repeatedly applied in the proof of the following theorem.

Theorem 2. (running time) *The running time of the CDS algorithm of Section 2 is $O(1.9407^n)$.*

Proof. Let $P(k)$ be the number of subproblems generated to solve an instance of size $k = k(S, D)$, where k is defined as in (1). For notational convenience we assume $P(k) = 1$ for $k \leq 0$.

Of course, if the algorithm halts, $P(k) = 1$. Now observe that when we apply the reduction rules (a)-(c), the size of the problem decreases by at least β:

$$P(k) \leq 1 + P(k - \beta). \tag{2}$$

Consider now the case when the algorithm branches. Note that, by (a), the candidate v selected is not a promise (and thus $\beta(v) = 1$). Following the algorithm, we distinguish different subcases:

(A) Suppose v dominates three free nodes w_1, w_2, and w_3 (and possibly more). By Fact 1,

$$\begin{aligned} P(k) \leq 1 &+ P(k - 1 - \Delta\alpha_{|w_1|} - \Delta\alpha_{|w_2|} - \Delta\alpha_{|w_3|}) \\ &+ P(k - 1 - \alpha_{|w_1|} - \alpha_{|w_2|} - \alpha_{|w_3|}). \end{aligned} \tag{3}$$

Now suppose v dominates an available node w, such that $N(w) \cap F \setminus N(v) = \emptyset$. Then when we select v, w is discarded by (c). Note that w cannot be a promise ($\beta(w) = 1$). By Fact 1 and the observation above,

$$P(k) \leq 1 + P(k - 1 - \Delta\alpha_{|w|}) + P(k - 1 - \alpha_{|w|} - 1). \tag{4}$$

From this point on we can assume that every available node w adjacent to the candidate v dominates at least one free node z not in $N(v)$. In the following we denote such free nodes by

$$Z(w) = N(w) \cap F \setminus N(v).$$

(B) Recall that v dominates a unique free node w, and U is the set of available neighbors of w, excluding $N[v]$. By Fact 1,

$$\begin{aligned} P(k) \leq 1 &+ P(k - 1 - \Delta\alpha_{|w|}) \\ &+ P(k - 1 - \alpha_{|w|} - \beta(w) - \sum_{z \in Z(w)} \delta_{w \in A} \cdot \alpha(z)) \\ &+ P(k - 1 - \alpha_{|w|} - \beta(w) - \sum_{z \in Z(w)} \delta_{w \in A} \cdot \Delta\alpha(z) - \sum_{u \in U} \beta(u)), \end{aligned} \tag{5}$$

where $\delta_{\mathcal{P}} = 1$ if predicate $\mathcal{P}$ is true, and $\delta_{\mathcal{P}} = 0$ otherwise.

Since v is not a promise, we have that $|U| = |w| - 1 \geq 1$. Moreover, by case (A), if w is available, it must dominate at least one free node z ($|Z(w)| \geq 1$). If w is not a promise, such a neighbor z must have frequency at least two.

It is worth to mention that there might be subproblems which are infeasible because we either select nodes which are discarded, or we discard nodes which are promises. In those cases we can replace the corresponding $P(k')$ with 1 in the recurrences above, since the algorithms halts on such subproblems. The same holds also in next cases.

(C) Recall that v dominates two free nodes w_1 and w_2, where U_i are the available neighbors of w_i, excluding $N[v]$. Moreover the w_i's are named such that, if w_2 is available (a promise), so is w_1. In particular this implies that, if w_1 is discarded, the same holds for w_2.

(C.1) In this case w_1 and w_2 are adjacent, w_1 is available and w_2 is discarded. Observe that, if $|w_1| = 2$, which implies $U_1 = \{u_{1,1}\}$, and $u_{1,1}$ is not a promise, then $u_{1,1}$ becomes a promise when we remove v. By this observation and Fact 1,

$$\begin{aligned} P(k) \leq 1 &+ P(k - 1 - \Delta\alpha_{|w_1|} - \Delta\alpha_{|w_2|} - \delta_{|w_1|=2}(\beta(u_{1,1}) - \beta)) \\ &+ P(k - 1 - \alpha_{|w_1|} - \alpha_{|w_2|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \alpha_{|z|}) \\ &+ P(k - 1 - \alpha_{|w_1|} - \alpha_{|w_2|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|} - \sum_{u \in U_1} \beta(u)). \end{aligned} \tag{6}$$

Note that $|U_1| = |w_1| - 1 \geq 1$.

(C.2) In this case w_1 and w_2 are adjacent and both available. Observe that, if $|w_1| = 2$ ($|w_1| = 2$) and w_2 (w_1) is not a promise, then w_2 (w_1) becomes a promise when we remove v. By this observation and Fact 1,

$$\begin{aligned}
P(k) \leq 1 &+ P(k-1-\sum_{i=1}^{2} \Delta\alpha_{|w_i|} - \sum_{i=1}^{2} \delta_{|w_i|=2} \cdot (\beta(w_i) - \beta)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \alpha(z)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha(z)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha(z) - \sum_{u \in U_1 \cup U_2} \beta(u)).
\end{aligned} \tag{7}$$

Note that it cannot be $|w_1| = |w_2| = 2$ since otherwise v would be a promise. Moreover $|U_1 \cup U_2| \geq \max\{|U_1|, |U_2|\} \geq \max\{|w_1| - 2, |w_2| - 2\}$.

(C.3) Recall that if w_1 and w_2 are adjacent, they are both discarded. In any case, $|U_1| = |w_1| - 1 \geq 1$ and $|U_2| = |w_2| - 1 \geq 1$. If $|w_1| = 2$, which implies $U_1 = \{u_{1,1}\}$, and $u_{1,1}$ is not a promise, $u_{1,1}$ becomes a promise when we remove v. A symmetric argument holds for w_2. By this observation and Fact 1,

$$\begin{aligned}
P(k) \leq 1 &+ P(k-1-\sum_{i=1}^{2} \Delta\alpha_{|w_i|} - \delta_{|w_1|=2 \text{ or } |w_2|=2} \max_{h}\{\beta(u_{h,1}) - \beta\}) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \beta(w_1) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \alpha_{|z|}) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|}) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|} - \sum_{u \in U_1} \beta(u)) \\
&+ P(k-1-\sum_{i=1}^{2} \alpha_{|w_i|} - \sum_{i=1}^{2} \beta(w_i) - \sum_{z \in Z(w_1)} \delta_{w_1 \in A} \cdot \Delta\alpha_{|z|} - \sum_{u \in U_2} \beta(u)).
\end{aligned} \tag{8}$$

Observe that, if w_1 is available (and thus w_1 and w_2 are not adjacent), by (A) w_1 must dominate at least one free node z: $|Z(w_1)| \geq 1$.

From recurrences (2)-(8), $P(k) \leq c^k \leq c^{(1+\alpha_3)n}$, where $c = c(\beta, \alpha_2, \alpha_3)$ is a quasi-convex function of the weights [7]. Thus the estimation of the running time reduces to choosing the weights minimizing $c^{1+\alpha_3}$. Note that it is sufficient to consider only a finite number of recurrences. This is because each recurrence R where the frequency of some node considered is larger than 5 is *dominated* by a recurrence R' where the same element has frequency 4, that is the upper bound on c given by R is not larger than the one given by R'. We numerically obtained $\beta = 0.5004$, $\alpha_2 = 0.0600$, and $\alpha_3 = 0.1215$, and thus the claimed running time $O(1.9407^n)$. □

4 An Exponential Lower Bound

Since the known tools to analyze the worst case running time of Branch and Reduce algorithms (including Measure and Conquer) do not provide tight upper bounds, it is natural to ask for lower bounds: A lower bound may give an idea of how far is the established upper bound from the real worst case running time.

Theorem 3. (lower bound) *The worst case running time of the CDS algorithm of Section 2 is $\Omega(4^{n/5}) = \Omega(1.3195^n)$.*

The proof of Theorem 3 is omitted here for lack of space.

References

1. R. Beigel and D. Eppstein. 3-coloring in time $O(1.3289^n)$. *Journal of Algorithms* 54:168–204, 2005.
2. J. Blum, M. Ding, A. Thaeler, and X. Cheng. Connected dominating set in sensor networks and MANETs. In *Handbook of combinatorial optimization. Supplement Vol. B*, pages 329–369. Springer, New York, 2005.
3. T. Brueggemann and W. Kern. An improved deterministic local search algorithm for 3-SAT. *Theoretical Computer Science* 329:303–313, 2004.
4. J. M. Byskov. Enumerating maximal independent sets with applications to graph colouring. *Operations Research Letters* 32:547–556, 2004.
5. E. Dantsin, A. Goerdt, E. A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science* 289:69–83, 2002.
6. D. Eppstein. The traveling salesman problem for cubic graphs. In *Workshop on Algorithms and Data Structures (WADS)*, pages 307–318, 2003.
7. D. Eppstein. Quasiconvex analysis of backtracking algorithms. Procedings of the *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 781–790, 2004.
8. F. V. Fomin, F. Grandoni, and D. Kratsch. Measure and Conquer: Domination - A Case Study, Proceedings of the *32nd International Colloquium on Automata, Languages and Programming (ICALP 2005)*, Springer LNCS vol. 3580, 2005, pp. 191–203.
9. F. V. Fomin, F. Grandoni, D. Kratsch. Some new techniques in design and analysis of exact (exponential) algorithms. *Bulletin of the EATCS* 87:47–77, 2005.
10. F. V. Fomin, F. Grandoni, D. Kratsch. Measure and Conquer: A simple $O(2^{0.288\,n})$ independent set algorithm. Procedings of the *17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 2006, pp. 18–25.
11. F. V. Fomin, D. Kratsch, and I. Todinca. Exact algorithms for treewidth and minimum fill-in. Proceedings of the *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Springer LNCS vol. 3142, 2004, pp.568–580.

12. F. V. Fomin, D. Kratsch, and G. J. Woeginger. Exact (exponential) algorithms for the dominating set problem. Proceedings of the *30th Workshop on Graph Theoretic Concepts in Computer Science (WG 2004)*, Springer LNCS vol. 3353, 2004, pp. 245-256.
13. M. R. Garey and D. S. Johnson. *Computers and Intractability. A Guide to the Theory of NP-Completeness.* Freemann, 1979.
14. F. Grandoni. A note on the complexity of minimum dominating set. *Journal of Discrete Algorithms*, 4(2):209–214, 2006.
15. S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. *Algorithmica*, 20(4):374–387, 1998.
16. A. Gupta, A. Kumar, and T. Roughgarden. Simpler and better approximation algorithms for network design. Proceedings of the *35th Annual ACM Symposium on Theory of Computing (STOC 2003)*, pages 365–372, New York, 2003. ACM.
17. J. Håstad. Clique is hard to approximate within $n^{1-\epsilon}$. *Acta Math.* 182 (1):105–142, 1999.
18. M. Held and R.M. Karp. A dynamic programming approach to sequencing problems. *Journal of SIAM*, pages 196–210, 1962.
19. R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity. *Journal of Computer and System Sciences* 63:512–530, 2001.
20. K. Iwama. Worst-case upper bounds for k-SAT. *Bulletin of the EATCS* 82:61–71, 2004.
21. D. Mölle, S. Richter, and P. Rossmanith. A faster algorithm for the steiner tree problem. Proceedings of the *23d Symposium on Theoretical Aspects of Computer Science (STACS 2006)*, Springer LNCS vol. 3884, 2006, pp. 561-570.
22. B. Randerath and I. Schiermeyer. Exact algorithms for MINIMUM DOMINATING SET. Technical Report, zaik-469, Zentrum für Angewandte Informatik Köln, April 2004.
23. I. Razgon. Exact computation of maximum induced forest. *Proceedings of the 10th Scandinavian Workshop on Algorithm Theory (SWAT 2006).* To appear.
24. J. M. Robson. Algorithms for maximum independent sets. *Journal of Algorithms* 7(3):425–440, 1986.
25. U. Schöning. Algorithmics in exponential time. Proceedings of the *22nd International Symposium on Theoretical Aspects of Computer Science (STACS 2005)*, Springer LNCS vol. 3404, 2005, pp. 36–43.
26. C. Swamy and A. Kumar. Primal-dual algorithms for connected facility location problems. *Algorithmica* 40(4):245–269, 2004.
27. R. Williams. A new algorithm for optimal constraint satisfaction and its implications. Proceedings of the *31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, Springer LNCS vol. 3142, 2004, pp. 1227–1237.
28. G. J. Woeginger. Exact algorithms for NP-hard problems: A survey. *Combinatorial Optimization – Eureka, You Shrink*, Springer LNCS vol. 2570, 2003, pp. 185–207.
29. G. J. Woeginger. Space and time complexity of exact algorithms: Some open problems. Proceedings of the *1st International Workshop on Parameterized and Exact Computation (IWPEC 2004)*, Springer LNCS vol. 3162, 2004, pp. 281–290.

Linear-Time Algorithms for Two Subtree-Comparison Problems on Phylogenetic Trees with Different Species

Sun-Yuan Hsieh

Department of Computer Science and Information Engineering,
National Cheng Kung University,
No 1. University Road, Tainan 70101, Taiwan
hsiehsy@mail.ncku.edu.tw

Abstract. Phylogenetic trees are an important tool to help in the understanding of relationships between objects that evolve through time, in particular molecular sequences. In this paper, we consider two subtree-comparison problems on phylogenetic trees. Given a set of k phylogenetic trees whose leaves are drawn from $\{1, 2, \ldots, n\}$ and the leaves for two arbitrary trees are not necessary the same, we first present a linear-time algorithm to final all maximal leaf-agreement subtrees. Based on this result, we also present a linear time algorithm to find maximal all-agreement isomorphic subtrees.

1 Introduction

One of the central problems in biology is to explain the evolutionary history of today's species and, in particular, how species relate to one another in terms of common ancestors. This is usually done by constructing trees, whose leaves represent present-day species and whose internal nodes represent hypothesized ancestors. These kinds of trees are called *phylogenetic trees* [19]. Phylogenetic trees are widely used for classifying hierarchical relations between different species [5, 10, 13]. Different methods of classification may lead to different trees. It is natural to try to resolve differing phylogenetic trees in a manner that will increase our confidence in the results.

There are quite a few phylogenetic inference methods, e.g. maximum parsimony, maximum likelihood, distance matrix fitting, subtrees consistency, and quarter based methods, proposed in the literature [8, 9, 11, 18, 20]. There were also many previous works for inferring the consensus tree from a profile of trees [1, 3, 5, 6]. Among them, many extensively studies focused on the *maximum agreement subtree problem* (MAST) [2]: Given a set of rooted trees whose leaves are drawn from the same set of items of size n, find the largest subset of items such that the portions of the trees restricted to the subset are isomorphic. It was shown that the problem is NP-hard even for three unbounded degree trees [1]. There were polynomial time algorithms for three or more bounded degree trees [1, 6], even though the time complexity is exponential in the bound for the degree. On the other hand, efficient algorithms for the MAST on two

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 164–175, 2006.

trees have been presented in the literature: Farach and Thorup [7] presented a $O(n^{1.5}\log n)$-time algorithm for two arbitrary degree trees. Cole *et al.* [4] showed that the MAST of two binary trees can be found in $O(n\log n)$ time, while the MAST of two degree d trees can be found in $O(min\{n\sqrt{d}\log^2 n, nd\log n\log d\})$ time.

Previously, it was rather common to compare phylogenetic trees on the same set of species with respect to different biological sequences or different genes. However, several biology experiments need to extract highly similar structures among a set of phylogenetic trees with different specie sets. In this paper, we consider two new subtree-comparison problems on a set of phylogenetic trees, which have applications to analyze the evolution and co-evolution genes clustering of genomic sequences [15, 17]. We begin with some definitions before proceeding to formally define the problems. Let T be a phylogenetic tree with at most n leaves such that each leaf is labelled with a distinct number in $\{1,2,\ldots,n\}$, and each internal node v has a unique label $label(v)$. The *subtree rooted at* v, denoted by $T[v]$, is the tree induced by descendants of v, rooted at v. Let $L(T[v]) = \{x|\ x \text{ is a leaf of } T[v]\}$. We also use $L(T)$ to denote the leaves of T. Two trees $T_i[v_i]$ and $T_j[v_j]$ are said to be *all-agreement isomorphic*, denoted by $T_i[v_i] \cong T_j[v_j]$, if the following conditions hold: Either $v_i \in T_i$ and $v_j \in T_j$ are two leaves with the same label; otherwise, $label(v_i) = label(v_j)$, and the children of v_i and the children of v_j can be put into one-to-one corresponding such that $T_i[v_{i_1}] \cong T_j[v_{j_1}]$, $T_i[v_{i_2}] \cong T_j[v_{j_2}]$, ..., and $T_i[v_{i_m}] \cong T_j[v_{j_m}]$, where $\{v_{i_1}, v_{i_2}, \ldots, v_{i_m}\}$ and $\{v_{j_1}, v_{j_2}, \ldots, v_{j_m}\}$ are the children of v_i and v_j, respectively. Given k rooted trees $T_1, T_2, \ldots, T_k$, where $L(T_i) \subseteq \{1,2,\ldots,n\}$ for $1 \le i \le k$, a k-tuple $(v_1, v_2, \ldots, v_k)$, where $v_i \in T_i$, is said to be *maximal leaf-agreement* if the following two conditions hold:

1. $L(T_1[v_1]) = L(T_2[v_2]) = \cdots = L(T_k[v_k])$.
2. There is no other sequence $(u_1, u_2, \ldots, u_k)$ satisfying $L(T_1[u_1]) = L(T_2[u_2]) = \cdots = L(T_k[u_k])$, and $T_i[v_i]$ is a subtree of $T_i[u_i]$ for $1 \le i \le k$.

If a given k-tuple only satisfies Condition 1, then we call it *leaf-agreement k-tuple.* In addition, a k-tuple is said to be *maximal all-agreement isomorphic* if the following two conditions hold:

1. $T_1[v_1] \cong T_2[v_2] \cong \cdots \cong T_k[v_k]$.
2. There is no other sequence $(u_1, u_2, \ldots, u_k)$ satisfying $T_1[u_1] \cong T_2[u_2] \cong \cdots \cong T_k[u_k]$, and $T_i[v_i]$ is a subtree of $T_i[u_i]$ for $1 \le i \le k$.

If a given k-tuple only satisfies Condition 1, then we call it *all-agreement isomorphic k-tuple.*

In this paper, we first show that the problem of finding all maximal leaf-agreement subtrees among the given k phylogenetic trees (that is, finding the set of maximal leaf-agreement k-tuples), can be solved in linear-time $O(kn)$. Based on this result, we also show that the problem of finding the set of maximal all-agreement isomorphic subtrees (that is, maximal all-agreement isomorphic k-tuples) can be solved in linear time $O(kn)$.

2 Preliminaries

Consider a rooted (unbounded degree) tree T. Let *root*(T) denote the root of T, and $V(T)$ denote the set of nodes of T. For a node v in T, any node y on the unique path from *root*(T) to v is called an *ancestor* of v. If y is an ancestor of v, then v is a *descendant* of y. Note that every node is both an ancestor and a descendant of itself. If y is an ancestor of v and $v \neq y$, then y is a *proper ancestor* of v and v is a *proper descendant* of y. If the last edge on the path from *root*(T) to a node x is (y, x), then y is the *parent* of x, and x is a *child* of y. If two nodes have the same parent, then they are *siblings*. In this paper, we assume that each internal node of T has at least two children; thus the total size of T is bounded by $O(n)$.

For a k-tuple $(v_1, v_2, \ldots, v_k)$, the ith position of the tuple is called the ith *dimension*. For convenience, $(v_1, v_2, \ldots, v_k)$ is said to be a *solution k-tuple* if it is a maximal leaf-agreement k-tuple or a maximal all-agreement isomorphic k-tuple. Given a set of k phylogenetic trees $T_1, T_2, \ldots, T_k$ with $L(T_i) \subseteq \{1, 2, \ldots, n\}$, a leaf $v \in T_i$, $1 \leq i \leq k$, is said to be *inactive* if each proper ancestor cannot appear in the ith dimension of a solution k-tuple. A node x is further said to be *unnecessary* if it is inactive or it is a proper ancestor of an inactive node. Let $\chi(T_i)$ be the set of inactive nodes of T_i. Also let $\chi = \bigcup_{1 \leq i \leq k} \chi(T_i)$.

Lemma 1. $\bigcup_{i=1}^{k} L(T_i) - \bigcap_{i=1}^{k} L(T_i) \subseteq \chi$.

Proof. If a node $x \notin \bigcap_{i=1}^{k} L(T_i)$, then the subtree rooted at each proper ancestor of x contains x as a leaf. By the definitions of leaf-agreement and all-agrement isomorphism, each proper ancestor of x cannot appear in the corresponding dimension of a solution k-tuple. □

For a node v in a rooted tree T, let $\pi(v)$ denote the unique path from v to the root of T. Moreover, we say a node $u \notin \pi(v)$ is *directly connected to* $\pi(v)$ if its parent is in $\pi(v)$.

Lemma 2. *Let $T_1, T_2, \ldots, T_k$ be a set of k phylogenetic trees with $L(T_i) \subseteq \{1, 2, \ldots, n\}$. If v is inactive of T_i, then any leaf which is directly connected to $\pi(v)$ is also inactive.*

Proof. Let l be a leaf node directly connected to $\pi(v)$ for an inactive node v. Since each proper ancestor u of l is also a proper ancestor of v, the subtree rooted at u contains both v and l. Therefore, u is not in any solution k-tuples. By the definition, l is inactive. □

Corollary 3. *Let $T_1, T_2, \ldots, T_k$ be a set of k phylogenetic trees with $L(T_i) \subseteq \{1, 2, \ldots, n\}$. Also let v is an inactive node of T_i and u is one of its sibling. If u is a leaf, then it is inactive.*

Proof. It directly follows from Lemma 2. □

The *least common ancestor* (*lca*) of two nodes u and v in a rooted tree T is the node w that is an ancestor of both u and v and that has the greatest depth in T, i.e., w is the first commonly encountered node by traversing paths from u and v to the root. For a set of nodes $U = \{u_1, u_2, \ldots, u_p\}$ in a rooted tree T, $lca_T(U)$ is used to denote an ancestor of $u_1, u_2, \ldots, u_p$ that has the greatest depth in T, i.e., $lca_T(U)$ is the first commonly encountered node by traversing paths from $u_1, u_2, \ldots, u_p$ to the root. The *level* of each node $v \in T$, denoted by $level(v)$, is the distance (number of edges) between v and the root.

3 Pruning Trees

The concept of our pruning algorithm is to delete those inactive nodes satisfying the condition of Lemma 1 together with their ancestors (without deleting all the inactive nodes). After pruning, all the resulting trees have the same leaves, we then further process these trees using algorithms described in Section 4 and Section 5.

Algorithm. RECONSTRUCTING_TREES
INPUT: a set of k phylogenetic trees $T_1, T_2, \ldots, T_k$ with $L(T_i) \subseteq \{1, 2, \ldots, n\}$;
OUTPUT: a set of k auxiliary trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ with $L(\mathcal{T}_1) = L(\mathcal{T}_2) = \cdots = L(\mathcal{T}_k)$.

```
1.  for each $T_i$, $1 \le i \le k$ do
    1.1 for each $v \notin \bigcap_{j=1}^{k} L(T_j)$ do
    1.2         delete $v$ together with those nodes in $\pi(v)$
    1.3 let $T_{i_1}, T_{i_2}, \ldots, T_{i_q}$ be the resulting rooted subtrees of $T_i$ after executing
        the above two steps
    1.4 if $q > 1$, then
    1.5         make a pseudo root $r_i$ and let it be the common parent of $T_{i_1}, T_{i_2}, \ldots, T_{i_q}$
    1.6         return the resulting tree $\mathcal{T}_i$
    1.7 else $\mathcal{T}_i \leftarrow T_i$      /* $q = 1$. Return the original tree. */
2.  Output $k$ auxiliary trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$.
```

For finding the desired k-tuples on $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$, we slightly modified the definitions of maximal leaf-agreement and all-agreement isomorphic. A k-tuple $(v_1, v_2, \ldots, v_k)$, where v_i is a non-root node in $\mathcal{T}_i$, is also said to be *maximal leaf-agreement on* $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$ if it satisfies Condition 1 of the original definition and Condition 2′: There is no other sequence $(u_1, u_2, \ldots, u_k)$, where $u_1, u_2, \ldots, u_k$ are all non-pseudo roots, such that $L(T_1[u_1]) = L(T_2[u_2]) = \cdots = L(T_k[u_k])$, and $T_i[v_i]$ is a subtree of $T_i[u_i]$ for $1 \le i \le k$. Similarly, we can also defined a *maximal all-agreement isomorphic k-tuple* on $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$.

Lemma 4. *After executing Algorithm* RECONSTRUCTING_TREES *for the input instance transformation, we show that finding maximal leaf-agreement k-tuples on $T_1, T_2, \ldots, T_k$ is equivalent to finding those on $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$.*

Proof. Suppose that $(v_1, v_2, \ldots, v_k)$ is a maximal leaf-agreement k-tuple on T_1, $T_2, \ldots, T_k$. Clearly, $L(T_1[v_1]) = L(T_2[v_2]) = \cdots = L(T_k[v_k])$ and each $T_i[v_i]$, $1 \leq i \leq k$, does not contain any inactive node in $\bigcup_{i=1}^{k} L(T_i) - \bigcap_{i=1}^{k} L(T_i)$. In executing Algorithm Reconstructing_Trees, if the parent of $root(T_i[v_i])$ is an unnecessary node, then $root(T_i[v_i])$ is directly connected to a pseudo root. In either case, $(v_1, v_2, \ldots, v_k)$ is also a maximal leaf-agreement k-tuple on $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$.

On the other hand, suppose that $(v_1, v_2, \ldots, v_k)$ is a maximal leaf-agreement k-tuple on $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$. Since each v_i for all $1 \leq i \leq k$, is not the pseudo root of $\mathcal{T}_i$, $\mathcal{T}_i[v_i]$ is also a subtree of T_i with $L(\mathcal{T}_1[v_1]) = L(\mathcal{T}_2[v_2]) = \cdots = L(\mathcal{T}_k[v_k]) = L(T_1[v_1]) = L(T_2[v_2]) = \cdots = L(T_k[v_k])$. Moreover, if there is a k-tuple $(u_1, u_2, \ldots, u_k)$, where $u_i \in T_i$, such that $L(T_1[u_1]) = L(T_2[u_2]) = \cdots = L(T_k[u_k])$ and $T_i[v_i]$ is a subtree of $T_i[u_i]$ for $1 \leq i \leq k$, then by Algorithm Reconstructing_Trees, $T_i[u_i]$ is also a subtree of $\mathcal{T}_i$ for $1 \leq i \leq k$, which contradicts to the fact that $(v_1, v_2, \ldots, v_k)$ is a maximal leaf-agreement k-tuple on $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$. Therefore, $(v_1, v_2, \ldots, v_k)$ is also maximal leaf-agreement k-tuple on $T_1, T_2, \ldots, T_k$. □

As with a proof similar to that of Lemma 4, we have the following result.

Lemma 5. *After executing Algorithm* Reconstructing_Trees *for the input instance transformation, we show that finding maximal all-agreement isomorphic k-tuples on $T_1, T_2, \ldots, T_k$ is equivalent to finding those on $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$.*

Lemma 6. *Algorithm* Reconstructing_Trees *can be implemented to run in $O(kn)$ time.*

Proof. We first describe our data structure. A rooted tree with unbounded degree is represented by the *left-child, right-sibling representation*[1]: Each node contains a parent pointer *par*, and *root*$[T]$ points to the root of tree T. Instead of having a pointer to each of its children, however, each node x has only two pointers, in which $left\text{-}child[x]$ points to the leftmost child of node x, and $right\text{-}sibling[x]$ points to the sibling of x immediately to the right. If node x has no children, then $left\text{-}child[x]$ =NIL (empty), and if node x is the rightmost child of its parent, then $right\text{-}child[x]$ =NIL. Note that this data structure use only $O(n)$ space.

We next describe the implementation of our algorithm. For implementing Steps 1.1–1.3, we first find all the leaves in $\bigcap_{i=1}^{k} L(T_i)$ using the auxiliary array $W[i, j]$, where $1 \leq i \leq n$ and $1 \leq j \leq k$, such that $W[i, j] = 1$ if tree T_j has a leaf labelled i (leaf i for short), and $W[i, j] = 0$ for otherwise. For each $1 \leq i \leq k$, we then compute the prefix-sum $\sum_{j=1}^{n} W[i, j]$. After this computation, if $W[i, j] = k$, then leaf i belongs to $\bigcap_{i=1}^{k} L(T_i)$. This implies that $\bigcup_{i=1}^{k} L(T_i) - \bigcap_{i=1}^{k} L(T_i)$ can be found in $O(kn)$ time. We then mark the corresponding unnecessary nodes on T_i: For each node $v \in \bigcup_{i=1}^{k} L(T_i) - \bigcap_{i=1}^{k} L(T_i)$, we mark each ancestor if it is unmark through pointer $par()$ from v to the root. This marking on T_i takes $O(n)$ time. Therefore, the unnecessary nodes of all the k trees can be found in

[1] For the remainder of this paper, a rooted tree is represented using this data structure for implementing algorithms.

$O(kn)$ time. We then removed them from each T_i and obtain several subtrees with the same time complexity.

We next describe the implementation of Step 1.4–1.7. During the above marking process, if there is an unmarked node with marked parent in T_i, then create the pseudo node r_i. Then, for each unmarked node x with marked parent, set $par(x)$ to r_i. This step can be implemented to run in $O(kn)$ time for all trees. Note that Step 2 can be easily implemented. Therefore, Algorithm RECONSTRUCTING_TREES runs in $O(kn)$ time. □

4 Maximal Leaf-Agreement Subtrees

Let $T_1, T_2, \ldots, T_k$ be a set of k phylogenetic trees. For two k-tuples $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ where $u_i, v_i \in T_i$, we say that $(u_1, u_2, \ldots, u_k)$ *properly contains* $(v_1, v_2, \ldots, v_k)$ if each $T_i[v_i]$ is a subtree of $T_i[u_i]$ for all $1 \leq i \leq k$. On the other hand, $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ are said to be *disjoint* if each of $\{u_i, v_i\}$ is not an ancestor of the other in T_i, for all $1 \leq i \leq k$.

Lemma 7. *Let $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ be two leaf-agreement k-tuples (i.e., $L(T[u_1]) = L(T[u_2]) = \cdots = L(T[u_k])$ and $L(T[v_1]) = L(T[v_2]) = \cdots = L(T[v_k])$). If there exists an integer $1 \leq i \leq k$ such that $L(T[u_i]) \cap L(T[v_i]) \neq \emptyset$, then either $L(T[u_i]) \subseteq L(T[v_i])$ or $L(T[v_i]) \subseteq L(T[u_i])$.*

Proof. Suppose, by contradiction, that $L(T[u_i]) \cap L(T[v_i]) \neq \emptyset$ and $L(T[u_i]) \nsubseteq L(T[v_i])$ and $L(T[v_i]) \nsubseteq L(T[u_i])$. Then, one of $\{u_i, v_i\}$ is not an ancestor of each other in T_i. Clearly, there is a unique path, denoted by P, in T_i connecting u_i and v_i via the root. On the other hand, by the assumption that $L(T[u_i]) \cap L(T[v_i]) \neq \emptyset$, there exists a leaf $l \in L(T[u_i]) \cap L(T[v_i])$. Then, there are two paths, one is from u_i to l and the other is form v_i to l; thus we can find another path Q different from P, which also connects u_i and v_i. This is a contradiction because paths P and Q form a cycle in T_i. □

Form Lemma 7, we know that if $L(T[u_j])$ and $L(T[v_j])$ have a non-empty intersection, then one contains the other.

Lemma 8. *If $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ are two leaf-agreement k-tuples, then there are no distinct $1 \leq i, j \leq k$ such that $L(T[u_i]) \cap L(T[v_i]) \neq \emptyset$, and $L(T[u_j])$ and $L(T[v_j])$ are disjoint.*

Proof. Suppose, by contradiction, that such i and j exist. Then, by Lemma 7, one of $\{T[u_i], T[v_i])\}$ is a subtree of the other. Without loss of generality, assume that $i < j$ and $T[v_i]$ is a subtree of $T[u_i]$. Then, $L(T[v_i]) \subseteq L(T[u_i])$. Since $L(T[u_1]) = L(T[u_2]) = \cdots = L(T[u_i]) = \cdots = L(T[u_j]) = \cdots = L(T[u_k])$, thus $L(T[v_j]) = L(T[v_i]) \subseteq L(T[u_i]) = L(T[u_j])$, which contradicts to the assumption that $L(T[u_j])$ and $L(T[v_j])$ are disjoint. □

Form Lemma 8, we know that if $L(T[u_j])$ and $L(T[v_j])$ have a non-empty intersection, then each all the other pairs $L(T[u_i])$ and $L(T[v_i])$ have non-empty intersection. By Lemma 8, we have the following immediate result.

Lemma 9. *Let $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ be two leaf-agreement k-tuples. If $L(T[v_j])$ and $L(T[u_j])$ are disjoint for some j, then $L(T[v_i])$ and $L(T[u_i])$ are disjoint for all $1 \leq i \leq k$.*

Lemma 10. *Let $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$ be two leaf-agreement k-tuples. If $L(T[v_j]) \subseteq L(T[u_j])$ for some j, then $L(T[v_i]) \subseteq L(T[u_i])$ for all $1 \leq i \leq k$.*

Proof. By the fact that $L(T[v_1]) = L(T[v_2]) = \cdots = L(T[v_j]) = \cdots = L(T[v_k]) \subseteq L(T[u_1]) = L(T[u_2]) = \cdots = L(T[u_j]) = \cdots = L(T[u_k])$, the result holds. □

Lemma 11. *For any two leaf-agreement k-tuples $(u_1, u_2, \ldots, u_k)$ and $(v_1, v_2, \ldots, v_k)$, both are disjoint or one properly contains the other.*

Proof. By Lemmas 7–10, the result holds. □

Given two phylogenetic trees T_1 and T_2 with the same leaves $L(T_1) = L(T_2)$, we define a function $f_{1,2} : V(T_1) \mapsto V(T_2)$ such that $f_{1,2}(x) = y = lca_{T_2}(L(T_1[x]))$. The following lemmas was shown in [16], which is useful to our algorithm.

Lemma 12. [16] *Let T_1 and T_2 be two phylogenetic trees with $L(T_1) = L(T_2)$, and x be a node in T_1 with children $x_1, x_2, \ldots, x_m$. Then,*

- $f_{1,2}(x) = lca_{T_2}(f_{1,2}(x_1), f_{1,2}(x_2), \ldots, f_{1,2}(x_m))$.
- $L(T_1[x]) = L(T_2[f_{1,2}(x)])$ *iff* $|L(T_1[x])| = |L(T_2[f_{1,2}(x)])|$.

In what follows, we first present a linear-time algorithm to find maximal leaf-agreement 2-tuples (pairs) on two trees, and then extend it to compute maximal leaf-agreement k-tuples on k trees. For convenience, we call each node v_i in a maximal leaf-agreement (all-agreement) k-tuple $(v_1, v_2, \ldots, v_k)$ as *critical node.*

```
Algorithm. MAXIMAL_PAIRS($T_1$,$T_2$)
INPUT: two phylogenetic trees $T_1$ and $T_2$ with $L(T_i) \subseteq \{1, 2, \ldots, n\}$;
OUTPUT: a set $M$ of maximal leaf-agreement pairs.

1. Apply Algorithm RECONSTRUCTING_TREES to output two auxiliary trees
   $\mathcal{T}_1$ and $\mathcal{T}_2$.
2. Compute a set of leaf-agreement pairs $S = \{(v_1, f_{1,2}(v_1))|\ |L(\mathcal{T}_1[v_1])| = |L(\mathcal{T}_2[f_{1,2}(v_1)])|\}$.
   By Lemma 12, this step can be carried out by a bottom-up evaluation of trees.
   For convenience, we call each node in the output pairs as a candidate.
3. Compute critical nodes of $\mathcal{T}_1$ as follows:
   3.1 $T \leftarrow \mathcal{T}_1$
   3.2 repeat processing each node $v$ in increasing order of $level(v)$ of the
       current tree $T$
   3.3     if $v$ is a candidate and it is not the pseudo root then
```

```
  3.4        mark $v$ and delete all the nodes of $T[v]$
  3.5        $T \leftarrow T - T[v]$
4. Output $M = \{(x, y)|\ (x, y) \in S$ and $x$ is marked$\}$.
```

Lemma 13. *Algorithm* MAXIMAL_PAIRS(T_1,T_2) *correctly computes all the maximal leaf-agreement pairs.*

Proof. It is clear that after executing Steps 1–2, all the candidates are collected in S. By Lemma 11, for any two pairs, either they are disjoint or one properly contains the other. This implies that $v \in T_1$ is critical iff it is a candidate and its ancestors are all not candidates. Clearly, after executing Steps 3, all the critical nodes in T_1 are obtained because each candidate which has a candidate ancestor is deleted in Step 3.4. Lemma 10 implies that if x is marked candidate, then $(x, y)(= (x, f_{1,2}(x)))$ is a maximal leaf-agreement pair. Therefore, all the maximal pairs can be found in Step 4 of the algorithm. □

Lemma 14. *Algorithm* MAXIMAL_PAIRS(T_1,T_2) *can be implemented to run in $O(n)$ time.*

Proof. By Lemma 6, Step 1 can be implemented to run in $O(n)$ time. By Lemma 12 and $O(1)$-time lca query [12], Step 2 can be implemented to run in $O(n)$ time by a bottom-up evaluation of trees.

Step 3 can be implemented as follows: $level(v)$ for all nodes v can be computed in $O(n)$ time using breadth-first-search. The increasing order of $level(v)$'s can be obtained by counting-sort to sort $n_1(= O(n))$ non-negative integers $level(v)$'s with range $[0, 1, \ldots, n_1]$ in $O(n)$ time. During travelling of the current tree T, when a unmarked candidate v is visited, we mark it and delete all the nodes of $T[v]$ in $O(p)$ time, where p is the number of nodes of $T[v]$. Since all the nodes of T_1 are processed at most twice, Step 3 can be implemented to run in $O(n)$ time.

Since $|S| = O(n)$, Step 4 can be implemented to run in $O(n)$ time. Therefore, Algorithm MAXIMAL_PAIRS(T_1,T_2) can be implemented to run in $O(n)$ time. □

Our result on two trees can be further extended to solve the problem on k trees as shown below.

Theorem 15. *Given a set of k phylogenetic trees $T_1, T_2, \ldots, T_k$ with $L(T_i) \in \{1, 2, \ldots, n\}$, all the maximal leaf-agreement k-tuples can be found in $O(kn)$ time.*

Proof. We describe our algorithm with the following steps: In Step 1, we apply algorithm RECONSTRUCTING_TREES to generate k auxiliary trees $\mathcal{T}_1, \mathcal{T}_2, \ldots, \mathcal{T}_k$. Note that $L(\mathcal{T}_1) = L(\mathcal{T}_2) = \cdots = L(\mathcal{T}_k)$. By Lemma 6, this step takes $O(kn)$ time.

In Step 2, we find all the leaf-agreement k-tuples $S = \{(v_1, v_2, \ldots, v_k)|\ v_1, v_2, \ldots, v_k$ are all non-seudo roots and $L(\mathcal{T}_1[v_1]) = L(\mathcal{T}_2[v_2]) = \cdots = L(\mathcal{T}_k[v_k])\}$ as follows: By utilizing Step 2 of Algorithm MAXIMAL_PAIRS($\mathcal{T}_1$,$\mathcal{T}_i$) for all $2 \leq i \leq k$, we can find $f_{1,2}(x), f_{1,3}(x), \ldots, f_{1,k}(x)$ for each non-pseudo root node $x \in \mathcal{T}_1$. Then, $(v, f_{1,2}(v), \ldots, f_{1,k}(v)) \in S$ iff $|L(\mathcal{T}_1[v])| = |L(\mathcal{T}_2[f_{1,2}(v)])| = \cdots = |L(\mathcal{T}_k[f_{1,k}(v)])|$. Since Step 2 of Algorithm MAXIMAL_PAIRS can be implemented to run in $O(n)$ time by the proof of Lemma 14, thus this step can be implemented to run in $O(kn)$ time.

In Step 3, we mark all the critical nodes of $\mathcal{T}_1$ using Step 3 of Algorithm Maximal_pairs. This step can be implemented to run in $O(n)$ time according to the proof of Lemma 14.

In Step 4, we output $\{(v, f_{1,2}(v), \ldots, f_{1,k}(v))| \ (v, f_{1,2}(v), \ldots, f_{1,k}(v)) \in S$ and v is marked$\}$. Since $|S| = O(n)$, thus this step can be implemented to run in $O(n)$ time. From the above analysis of Steps 1–4, the overall time complexity is $O(kn)$. As with a similar argument to show Lemma 13, it is not difficult to verify the correctness of the algorithm. Therefore, the result holds. □

5 Maximal All-Agreement Isomorphic Subtree

Given a set of k phylogenetic trees, we aim at computing all the maximal all-agreement isomorphic k-tuples by utilizing the method presented in Section 4. We begin solving the problem on $k = 2$. For two nodes $x \in T_1$ and $y \in T_2$, it is clear that (x, y) is a maximal all-agreement isomorphic pair implies that (x, y) is a maximal leaf-agreement pair. By the definition of $T_1 \cong T_2$, we have the following lemma.

Lemma 16. *Let T_1 and T_2 be two phylogenetic trees with $L(T_1) = L(T_2)$, and let v be a node in T_1 with the children $\{v_1, v_2, \ldots, v_m\}$. If label$(v)$ = label$(f_{1,2}(v))$ and $T_1[v_i] \cong T_2[f_{1,2}(v_i)]$ for all $1 \leq i \leq m$, and $f_{1,2}(v_i)$ are all the children of $f_{1,2}(v)$, then $T_1[v] \cong T_2[f_{1,2}(v)]$*

We now present a linear-time algorithm to find maximal all-agreement isomorphic pairs on two trees.

Algorithm. Isomorphic_pairs(T_1,T_2)
Input: two phylogenetic trees T_1 and T_2 with $L(T_i) \subseteq \{1, 2, \ldots, n\}$;
Output: a set I of maximal all-agreement isomorphic pairs.

```
1. Apply Algorithm Maximal_pairs($T_1$,$T_2$) to output a set of maximal leaf-
   agreement pairs $M = \{(x, y)\}$.
2. Test whether $T_1[x]$ and $T_2[y]$ are all-agrement isomorphic for each pair
   $(x, y) \in M$:
   2.1 for each pair $(x, y) \in M$ do
   2.2    for each node $v$ in non-increasing order of $level(v)$ of $T_1[x]$ do
   2.3      if $v$ is a leaf then
   2.4        mark $v$
   2.5      else /* $v$ is an internal node of $T_1[x]$ */
   2.6        let $\{v_1, v_2, \ldots, v_m\}$ be the children of $v$
   2.7          if $label(v) = label(f_{1,2}(v))$ and all $v_i$'s are marked and $f_{1,2}(v_i)$'s
                   are all the children of $f_{1,2}(v)$ then
```

```
  2.8          mark $v$
  2.9        **else** ignore this pair
3. Output $I = \{(x, y) | \ (x, y) \in M$ and $x$ is marked$\}$.
```

Lemma 17. *Algorithm* ISOMORPHIC_PAIRS(T_1,T_2) *correctly computes all the maximal isomorphic pairs.*

Proof. For two nodes $x \in T_1$ and $y \in T_2$, $T_1[x]$ and $T_2[y]$ are maximal all-agreement isomorphic implies that $L(T_1[x]) = L(T_2[y])$ and $T_1[x]$ and $T_2[y]$ are not subtrees of another two leaf-agreement trees $T_1[x']$ and $T_1[y']$, respectively. Thus we only need to verify that for each maximal leaf-agreement pair (x, y), whether $T_1[x] \cong T_2[y]$ (Steps 1–2). We define the *height* of a node v, denoted by *height*(v), in a tree is the number of edges on the longest simple downward path from the node to a leaf. We next show by induction that Steps 2.1–2.9 correctly verify whether two subtrees are all-agreement isomorphic:

Claim 1. During executing Steps 2.1–2.9, if a node $v \in T_1$ is marked, then $T_1[v] \cong T_2[f_{1,2}(v)]$.

Proof of the Claim: We prove the claim by induction on *height*(v).

BASIS: *height*$(v) = 0$. Then v is a leaf. The result holds because $v \in T_1$ and $f_{1,2}(v) \in T_2$ are two one-node trees with the same label. Assume the result holds on *height*$(v) = h > 0$.

INDUCTION STEP: Now we consider *height*$(v) = h + 1$. Let $v_1, v_2, \ldots, v_m$ be the children of v. By Steps 2.7–2.8, v is marked provided that $v_1, v_2, \ldots, v_m$ are all marked. By the induction hypothesis, $T_1[v_1] \cong T_2[f_{1,2}(v_1)], T_1[v_2] \cong T_2[f_{1,2}(v_2)], \ldots, T_1[v_m] \cong T_2[f_{1,2}(v_m)]$ hold. Moreover, *label*$(v) =$ *label* $(f_{1,2}(v))$ and $f_{1,2}(v_i)$'s for all $1 \leq i \leq m$ are the children of $f_{1,2}(v)$ (verified by Step 2.7). By Lemma 16, $T_1[v] \cong T_2[f_{1,2}(v)]$. The result of this claim holds.

Therefore, for a maximal leaf-agreement pair (x, y), if x is marked, then $T_1[x] \cong T_2[f_{1,2}(x)](= T_2[y])$, then our algorithm outputs it by Step 3. On the other hand, if x is unmarked, then there exists some descendant, say v, of x such that a subtree $T_1[v]$ of $T_1[x]$ is not all-agreement isomorphic to a subtree $T_2[f_{1,2}(v)]$ of $T_2[y]$. In such a case, Step 2.9 will ignore the pair (x, y). Thus the algorithm is correct. □

We next analyze the time-complexity of Algorithm ISOMORPHIC_PAIRS(T_1,T_2).

Lemma 18. *Algorithm* ISOMORPHIC_PAIRS(T_1,T_2) *can be implemented to run in $O(n)$ time.*

Proof. By Lemma 14, Step 1 can be implemented to run in $O(n)$ time.

Before proceeding to analyze Step 2, we first note that *level*(v)'s for all nodes v in T_1 can be computed in $O(n)$ time. Assume that the number of nodes of T_1 is $n_1 = O(n)$. By utilizing the counting sort to sort n_1 non-negative integers with range $[0, 1, \ldots, n_1]$, the decreasing order of *level*(v)'s can be computed in $O(n)$

time. We also note that $f_{1,2}(v)$'s for all nodes v in T_1 can also be computed in $O(n)$ time by the proof of Lemma 14. We assume that $label(f_{1,2}(v))$ for all nodes $v \in T_1$ are stored in an auxiliary table during the computation of Algorithm MAXIMAL_PAIRS. Moreover, by utilizing the representation of T_2, we can compute, in advance, the number of children of each node in T_2 in $O(n_2+m_2) = O(n)$ time, where n_2 and m_2 are the number of nodes and edges of T_2, respectively. Then, Step 2 can be implemented as follows: We visit all the nodes of T_1 in non-increasing order of $level()$'s in $O(n)$ time, and process each node v in $O(1)$ time depending on the following two cases:

CASE 1: v is a leaf. Mark v.

CASE 2: v is an internal node. Assume that $\{v_1, v_2, \ldots, v_m\}$ be the children of v. We mark v if $label(v) = label(f_{1,2}(v))$ and all v_i's are marked and $f_{1,2}(v_i)$ are all the children of $f_{1,2}(v)$. This checking can be carried out in $O(deg(v))$ time, where $deg(v)$ is the number of children of v in T_1.

After travelling T_1 together with the above operations, Step 2 can be implemented to run in $O(n_1 + m_1) = O(n)$ time.

Because $|M| = O(n)$, Step 3 can be easily implemented to run in $O(n)$ time. Therefore, Algorithm ISOMORPHIC_PAIRS(T_1,T_2) can be implemented to run in $O(n)$ time. □

Our result on two trees can be further extended to solve the problem on k trees as shown below.

Theorem 19. *Given a set of k phylogenetic trees $T_1, T_2, \ldots, T_k$ with $L(T_i) \subseteq \{1, 2, \ldots, n\}$, the maximal all-agreement isomorphic k-tuples can be found in $O(kn)$ time.*

Proof. Our algorithm is described as follows: First, we generate the maximal leaf-agreement k-tuples in $O(kn)$ time by Theorem 15.

Next, for each maximal leaf-agreement k-tuple $(v_1, v_2, \ldots, v_k)$, we can verify whether $T_1[v_1] \cong T_2[v_2] \cong \cdots \cong T_k[v_k]$ as follows: Note that $T_1[v_1] \cong T_2[v_2] \cong \cdots \cong T_k[v_k]$ iff $T_1[v_1] \cong T_2[v_2], T_1[v_1] \cong T_3[v_3], \ldots, T_1[v_1] \cong T_k[v_k]$. That is $(v_1, v_2, \ldots, v_k)$ is maximal all-agreement isomorphic iff $(v_1, v_2), (v_1, v_3), \ldots,$ (v_1, v_k) are all maximal all-agreement isomorphic pairs. By executing Step 2 of Algorithm ISOMORPHIC_PAIRS(T_1,T_i) for all $2 \leq i \leq k$, we can verify whether (v_1, v_i)'s are maximal all-agreement isomorphic pairs. We note that the number of the maximal all-agreement isomorphic k-tuples is bounded by $O(n)$. As with a similar method to implement Step 2 of Algorithm ISOMORPHIC_PAIRS(,), the maximal all-agreement isomorphic k-tuples can be obtained in $O(kn)$ time. □

References

1. A. Amir and D. Keselman, "Maximum agreement subtree in a set of evolutionary trees: Metrics and efficient algorithms," *SIAM Journal on Computing*, 26(6):1656-1669, 1997

2. P. Bonizzoni, G. Della Vedova, and G. Mauri, "Approximating the maximum isomorphic agreement subtree is hard," In R. Giancarlo and D. Sankoff editors, *Proceedings of the11th Annual Symposium on Combinatorial Pattern Matching*, Lecture Notes in Computer Science 1848, pages 119-128, Montréal, Canada, 2000.
3. D. Bryant, *Building Trees, Hunting for Trees, and Comparing Trees*, PhD thesis, University of Canterbury, Christchurch, New Zealand, 1997.
4. R. Cole, M. Farach, R. Hariharan, T. Przytycka, and M. Thorup, "An $O(nlogn)$ algorithm for the maximum agreement subtree problem for binary trees," *SIAM Journal on Computing*, 30(5):1385-1404, 2002.
5. W. H. E. Day, "Optimal algorithms for comparing trees with labelled leaves," *Journal of Classification*, 2:7-28, 1985.
6. M. Farach, T.M. Przytycka, and M. Thorup, "On the agreement of many trees," *Information Processing Letters*, 55(6):297-301, 1995.
7. M. Farach and M. Thorup, "Sparse dynamic programming for evolutionary-tree comparison," *SIAM Journal on Computing*, 26(1):210-230, 1997.
8. J. Felsenstein, "Numerical methods for inferring evolutionary trees," *Quarterly Review on Biology*, 57(4):379-404, 1982.
9. W. M. Fitch, "Toward defining the course of evolution: minimal change for a specific tree topology," *Systematic Zoology*, 20:406-441, 1971.
10. A. D. Gordon, "On the assessment and comparison of classifications," in *Analyse de Donnēes et Informatique, R. Tomassone, ed.*, INRIA, pp. 149–160, 1980.
11. D. Gusfield, "Efficient algorithms for inferring evolutionary trees," *Networks*, 21:19-28, 1991.
12. D. Harel and R. E. Tarjan, "Fast algorithms for finding nearest common ancestors," *SIAM Journal on Computing*, 13(2):338-355, 1984.
13. J. A. Hartigan, *Clustering Algorithms*, John Wiley, 1975.
14. J. Hein, T. Jiang, L. Wang, and K. Zhang, "On the complexity of comparing evolutionary trees," *Discrete Applied Mathematics*, 71:153-169, 1996.
15. J. A. Hoch and T. J. Silhavy, *Two-Component Signal Transduction*, ASM press, 1995.
16. Y. L. Lin and T. S. Hsu, "Efficient algorithms for descendent subtrees comparison of phylogenetic trees with applications to co-evolutionary classifications in bacterial genome," in *Proceedings of the 16th Annual International Symposium on Algorithms and Computation* (ISAAC), Lecture Notes in Computer Science 2906, pp. 339-351, 2003.
17. A. Rodrigue, Y. Quentin, A. Lazdunski, V. M/'ejean, and M. Foglino, "Two-component systems in pseudomonas aeruginosa: why so many?," *Trends Microbiol.* 8:498–504, 2000.
18. N. Saitou and M. Nei, "The neighbor-joining method: a new method for reconstructing phylogenetic trees," *Molecular Biology Evolution*, 4:406-425, 1987.
19. J. C. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*, PWS publishing company, 1997.
20. K. Strimmer and A. von Haeseler, "Quartet puzzling: a quartet maximum-likelihood method for reconstructing tree topologies," *Molecular Biology and Evolution*, 13(7):964-969, 1996.

Computationally Sound Symbolic Secrecy in the Presence of Hash Functions*

Véronique Cortier[1], Steve Kremer[2], Ralf Küsters[3], and Bogdan Warinschi[4]

[1] Loria, CNRS & INRIA project Cassis, France
[2] LSV, CNRS & ENS Cachan & INRIA project Secsi, France
[3] ETH Zurich, Switzerland
[4] Loria, Univerité Henri Poincaré & INRIA project Cassis, France

Abstract. The standard symbolic, deducibility-based notions of secrecy are in general insufficient from a cryptographic point of view, especially in presence of hash functions. In this paper we devise and motivate a more appropriate secrecy criterion which exactly captures a standard cryptographic notion of secrecy for protocols involving public-key enryption and hash functions: protocols that satisfy it are computationally secure while any violation of our criterion directly leads to an attack. Furthermore, we prove that our criterion is decidable via an NP decision procedure. Our results hold for standard security notions for encryption and hash functions modeled as random oracles.

1 Introduction

Two distinct kinds of models have been developed for the rigorous design and analysis of cryptographic protocols: the so-called Dolev-Yao, symbolic, or formal models on the one hand and the cryptographic, computational, or concrete models on the other hand. In symbolic models messages are considered as formal terms and the adversary can manipulate these terms based on a fixed set of operations. The main advantage of the symbolic approach is its relative simplicity which makes it amenable to automated analysis tools (see, e.g., [6,15]). In cryptographic models, messages are actual bit strings and the adversary is an arbitrary probabilistic polynomial-time (ppt) Turing machine. While proofs in this kind of models yield strong security guarantees, the proofs are often quite involved and only rarely suitable for automation (see, e.g., [11,5]).

Starting with the seminal work of Abadi and Rogaway [1], a significant amount of research has been directed at bridging the gap between the two approaches. The goal is to obtain the best of both worlds: simple, automated security proofs that entail strong security guarantees. The typical approach is to show that the executions of the computational adversaries correspond to executions of the symbolic adversaries, and then use this result to show how to translate security notions from the symbolic world to the computational world.

For some security notions like integrity and authentication, the derivation of computational guarantees out of symbolic ones can be done with relative simplicity [3,14]. In contrast, analogous results for the basic notion of secrecy proved significantly more

* Work partly supported by ACI Jeunes Chercheurs JC9005 and ARA SSIA Formacrypt.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 176–187, 2006.

elusive and have appeared only recently [4,10,12,7]. The apparent reason for this situation is the striking difference between the definitional ideas used in the two different models. Symbolic secrecy typically states that the adversary cannot deduce the entire secret from the messages it gathers in an execution. On the other hand, computational secrecy requires that not only the secret, but also no partial information is leaked to the adversary. A typical formulation that is used requires the adversary to distinguish between the secret and a completely unrelated alternative.

OUR CONTRIBUTIONS. In this paper we investigate soundness results for symbolic secrecy in the presence of hash functions. One of the main motivations for considering hash functions, which have not been considered in the aforementioned results[1], is that they present a new challenge in linking symbolic and cryptographic secrecy: Unlike ciphertexts, hashes have to be publicly verifiable, i.e., any third party can verify if a value h is the hash value corresponding to a given message m. This implies that a simple minded extension of previous results on symbolic and computational secrecy fails. Assume, for example, that in some protocol the hash $h = h(s)$ of some secret s is sent in clear over the network. Then, while virtually all symbolic models would conclude that s remains secret (and this is also a naive assumption often made in practice), a trivial attack works in computational models: given s, s' and h, compare h with $h(s)$ and $h(s')$, and therefore recover s. Similar verifiability properties also occur in other settings, e.g. digital signatures which do not reveal the message signed.

In this paper we propose a new symbolic definition for nonce secrecy in protocols that use party identities, nonces, hash functions, and public key encryption. The definition that we give is based on the intuitively appealing concept of patterns [1].

The central aspect of our criterion is that it captures precisely security in the computational world in the sense that it is both sound and complete. More specifically, nonces that are secret according to our *symbolic* criterion are also secret according to a standard *computational* definition. Furthermore, there exist successful attacks against the secrecy of any nonce that does not satisfy our definition. Our theorems hold for protocols implemented with encryption schemes that satisfy standard notions of security, and for hash functions modeled as random oracles. In the proofs we combine different techniques from cryptography and make direct use of a (non-trivial) extension of the mapping theorem of [14] to hash functions.

Our second important result is to prove the decidability of our symbolic secrecy criterion (w.r.t. a bounded number of sessions). This is a crucial result that enables the automatic verification of computational secrecy for nonces. We give an NP-decision procedure based on constraint solving, a technique that is suitable for practical implementations [2]. While the constraint solving technique is standard in automatic protocol analysis, we had to adapt it for our symbolic secrecy criterion: For the standard deducibility-based secrecy definition it suffices to transform constraint systems until one obtains a so-called simple form. However, for our symbolic secrecy criterion further transformations might be required in order for the procedure to be complete. Identifying a sufficient set of such transformations and proving that they are sufficient turned out to be non-trivial.

[1] One exception is [12] where hash functions are allowed, but only as randomness extractors.

Related Work. The papers that are immediately related to our work are those of Cortier and Warinschi [10], Backes and Pfitzmann [4], and Canetti and Herzog [7], who study computationally sound secrecy properties, as well as the paper by Janvier et al. [13], that presents a soundness result in the presence of hash functions. In this context, our work is the first to tackle computationally sound secrecy in the presence of hashes. We study the translation of symbolic secrecy into a computational version in a setting closely related to that in [10]. However, the use of hashes requires, as explained above, new notions and non-trivial extensions of the results proved there. In [13], Janvier et al. present a soundness result that differs however from this one. On the one hand they do not consider computational secrecy of nonces sent under hash functions. On the other hand, they present a new security criterion for hash functions, which is not the random oracle, although no implementation of a hash function satisfying their criterion is currently known. The work in [4] and [7] is concerned with secrecy properties of key-exchange protocols in the context of simulation-based security, and hence, they study different computational settings. Interestingly, the symbolic criterion used in [7] is also formalized using patterns, but their use is unrelated to ours. None of the mentioned works considers decidability issues.

Paper outline. In the following section, we introduce the symbolic and computational models. Our symbolic secrecy criterion is developed in Section 3. We state and prove the soundness and completeness of this criterion w.r.t. computational secrecy in Section 4, and prove its decidability in Section 5. Details about the models and proofs can be found in [9].

2 The Symbolic and Concrete Protocol and Intruder Models

2.1 The Symbolic Model

We define (symbolic) messages and terms, how honest agents and the (Dolev-Yao-style) intruder can derive messages from a set of messages, and how protocols are specified.

Messages and Terms. To define messages, we consider an infinite set A of agent identities, infinite sets Nonce_{ag}, Nonce_{adv}, Rand_{ag}, and Rand_{adv} (nonces and random coins generated by the agents and the adversary, respectively), and an infinite set $\mathsf{Garbage}$ representing garbage messages. All of these sets are assumed to be pairwise disjoint. We set $\mathsf{Nonce} = \mathsf{Nonce}_{ag} \cup \mathsf{Nonce}_{adv}$ and $\mathsf{Rand} = \mathsf{Rand}_{ag} \cup \mathsf{Rand}_{adv}$.

The set of messages M (w.r.t. A, Nonce, and Rand) is defined by the following grammar: $\mathsf{M} ::= \mathsf{A} \mid \mathsf{Nonce} \mid \mathsf{ek}(\mathsf{A}) \mid \mathsf{dk}(\mathsf{A}) \mid \langle \mathsf{M}, \mathsf{M} \rangle \mid \{\mathsf{M}\}^{\mathsf{Rand}}_{\mathsf{ek}(\mathsf{A})} \mid \mathsf{h}(\mathsf{M}) \mid \mathsf{Garbage}$ where $\mathsf{ek}(a)$ and $\mathsf{dk}(a)$ with $a \in \mathsf{A}$ denote the public and private key of a, respectively, $\langle m, m' \rangle$ denotes pairing of m and m', $\{m\}^{r}_{\mathsf{ek}(a)}$ denotes the message m encrypted with $\mathsf{ek}(a)$ using the random coins r, and $h(m)$ is the hash of m. We define the following subsets of M: EKey, DKey, $\mathsf{Ciphertext}$, Hash, and Pair are the sets of all messages starting with $\mathsf{ek}(\cdot)$, $\mathsf{dk}(\cdot)$, $\{\cdot\}^{\cdot}_{\cdot}$, $\mathsf{h}(\cdot)$, and $\langle \cdot, \cdot \rangle$, respectively. We sometimes refer to the sets introduced above as types.

We assume an infinite set of typed variables X where the types are as above and for a variable of a certain type only messages of this type may be substituted. In particular, we assume variables A_i, $i \in \{1, \ldots, k\}$, for agent identities and variables $X^j_{A_i}$, $L^j_{A_i}$

$j \in \mathbb{N}$, for fresh nonces and random coins generated by A_i. The set of terms $\mathsf{T}(\mathsf{X})$ over X is defined analogously to the set of messages.

DERIVING MESSAGES. terms. The set of terms that can be derived from ϕ is defined by the deduction rules given in Figure 1. We write $\phi \vdash t$ to say that t can be derived from ϕ. For example, $\{\langle \mathsf{dk}(a), \{c\}^r_{\mathsf{ek}(a)}\rangle\} \cup \mathsf{A} \vdash \{c\}^{r'}_{\mathsf{ek}(b)}$ where $b \in \mathsf{A}$ and $r' \in \mathsf{Rand}_{adv}$.

$$\frac{}{\phi \vdash m}\ m \in \phi \qquad \frac{\phi \vdash b}{\phi \vdash \mathsf{ek}(b)}\ b \in \mathsf{A} \cup \mathsf{X}.a \qquad \frac{\phi \vdash m_1 \quad \phi \vdash m_2}{\phi \vdash \langle m_1, m_2 \rangle} \qquad \frac{\phi \vdash \langle m_1, m_2 \rangle}{\phi \vdash m_i}\ i \in \{1,2\}$$

$$\frac{\phi \vdash \mathsf{ek}(b), \phi \vdash m}{\phi \vdash \{m\}^r_{\mathsf{ek}(b)}}\ r \in \mathsf{Rand}_{adv} \qquad \frac{\phi \vdash \{m\}^r_{\mathsf{ek}(b)} \quad \phi \vdash \mathsf{dk}(b)}{\phi \vdash m} \qquad \frac{\phi \vdash m}{\phi \vdash \mathsf{h}(m)}$$

Fig. 1. Deduction rules

PROTOCOLS. Roles are usually specified by a sequence of input/output actions. In order to model branching protocols, the *roles* we consider are ordered edge-labeled finite trees where every edge is labeled by an *agent rule* (l, r), where $l, r \in \mathsf{T}(\mathsf{X})$ are messages with variables, and certain syntactic conditions are satisfied such that the actions can actually be carried out (in a computational interpretation). A *k-party protocol* is a mapping $\Pi : [k] \to \mathsf{Roles}$ where $[k] = \{1, \ldots, k\}$ and Roles denotes the set of roles.

SYMBOLIC EXECUTION OF A PROTOCOL. The symbolic execution of a k-party protocol is modeled as a finite sequence of global states. A *global state* is a triple $(\mathsf{SId}, f, \varphi)$ where φ is a finite set of messages (the *current intruder knowledge*), SId is a finite set of session ids, and f maps every session id in SId to the current state of the corresponding session. This state is called the *local state* and is of the form $(i, \sigma, p, (a_1, a_2, \ldots, a_k))$ where $i \in [k]$ is the index of the role that is executed in this session, σ is a substitution whose domain is a subset of the variables occurring in $\Pi(i)$ (i.e., σ determines the messages assigned to variables so far in the current session), p is a node of $\Pi(i)$ and determines at what node the agent currently stands, and $(a_1, a_2, \ldots, a_k) \in \mathsf{A}^k$ is the tuple of names of the agents that are involved in the session, where a_i is the agent carrying out the current session (supposedly with the mentioned agents a_j, $j \neq i$). The initial state is $q_I = (\emptyset, \emptyset, \mathsf{A} \cup \mathsf{EKey} \cup \mathsf{Nonce}_{adv})$, i.e., the intruder knows all names and public keys of agents as well as the infinite set of intruder nonces.

We allow three kinds of transitions between global states.

- The adversary corrupts a set of parties and thereby learns the private keys of the agents: $q_I \xrightarrow{\mathbf{corrupt}(a_1, \ldots, a_l)} (\emptyset, \emptyset, \mathsf{A} \cup \mathsf{EKey} \cup \{\mathsf{dk}(a_j) \mid 1 \leq j \leq l\})$. Note that this transition can only be applied at the beginning (static corruption).
- The adversary can initiate new sessions: $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{new}(i, a_1, \ldots, a_k)} (\mathsf{SId}', f', \varphi)$ where SId' and f' are defined as follows. Let $\mathsf{sid} = |\mathsf{SId}| + 1$ be the session identifier of the new session where $|\mathsf{SId}|$ denotes the cardinality of SId. We define $\mathsf{SId}' = \mathsf{SId} \cup \{\mathsf{sid}\}$. The function f' is defined as: $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \in \mathsf{SId}$

and $f'(\mathsf{sid}) = (i, \sigma, \varepsilon, (a_1, \ldots, a_k))$ where ϵ denotes the root of the role tree and $\sigma(A_j) = a_j$ for $1 \leq j \leq k$ and $\sigma(X^j_{A_i}) = n^{a_i,j,s}$, $\sigma(L^j_{A_i}) = l^{a_i,j,s}$ for $j \in \mathbb{N}$.

- The adversary can send messages: $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}, f', \varphi')$ where $\mathsf{sid} \in \mathsf{SId}$, $m \in \mathsf{M}$, and φ' and f' are defined as follows. We define $f'(\mathsf{sid}') = f(\mathsf{sid}')$ for every $\mathsf{sid}' \neq \mathsf{sid}$. Suppose that $f(\mathsf{sid}) = (i, \sigma, p, (a_1, \ldots, a_k))$ and $(l_1, r_1), \ldots, (l_h, r_h)$ are the labels of edges leaving p (in this order). We distinguish two cases:
  - there does not exist a j such that m and $l_j\sigma$ match. Then, we define $f'(\mathsf{sid}) = f(\mathsf{sid})$ and $\varphi' = \varphi$ (the state remains unchanged);
  - else, let j be minimal s. t. m and $l_j\sigma$ match. Let θ be the matcher, i.e., $m = (l_j\sigma)\theta$. We define $f'(\mathsf{sid}) = (i, \sigma \cup \theta, pj, (a_1, \ldots, a_k))$ and $\varphi' = \varphi \cup \{(r_j\tau_{a_i,\mathsf{sid}})\sigma\theta\}$.

A finite sequence of global states is called a *symbolic execution trace* (for a protocol Π) if it starts with the initial global state q_I and two consecutive global states in this sequence are connected via one of the above transitions. We say that a trace is *valid* if every send transition $(\mathsf{SId}, f, \varphi) \xrightarrow{\mathbf{send}(\mathsf{sid},m)} (\mathsf{SId}, f', \varphi')$ verifies that the adversary could actually deduce m, that is $\varphi \vdash m$. The set of valid symbolic execution traces (for a protocol Π) is denoted by $\mathsf{Exec}^s(\Pi)$. The set of valid set of messages is defined by $\mathsf{Msg}^s(\Pi) = \{\varphi \mid (\mathsf{SId}, f, \varphi)$ is the last state of a valid execution trace$\}$.

2.2 The Concrete Model

The concrete model is defined w.r.t. an encryption scheme $\mathcal{AE} = (\mathsf{K_e}, \mathsf{Enc}, \mathsf{Dec})$, which we now fix once and for all. Hashing is modeled by the random oracle.

CONCRETE MESSAGES. *Concrete messages* are bit strings which carry type informationwhich can be efficiently computed. In bit strings of type Pair, the two components can be efficiently retrievedand strings of type Ciphertext carry the public key that supposedly was used to encrypt the plaintext. The set of bit strings is denoted by $\mathcal{C}^\eta$. This set depends on the security parameter η as this parameter determines the length of agent names, nonces, and keys. Substitutions now map variables (of some type) to concrete messages (of the same type).

CONCRETE EXECUTION OF A PROTOCOL. A *concrete global state* is a 4-tuple $(\mathsf{SId}, f, \varphi, \mathcal{H})$ where φ is a finite set of bit strings, SId is a finite set of session ids, and f maps every session id in SId to the current state of the corresponding session (the concrete local states). A *concrete local state* is defined just as a symbolic one, except that variables are now mapped to bit strings and agent names are also bit strings. The fourth component carries the state of the random oracle: $\mathcal{H}$ is a set of couples (m, h) where m is a bit string and h its corresponding hash value. A protocol is executed by running a ppt Turing machine, the (concrete) adversary, which may make queries corresponding to the transitions in the symbolic model. We allow four kinds of transitions between global states, which we will refer to by *corrupt*, *new*, *send transitions*, and *hash queries*. The semantics of the first three queries is defined by analogy with the formal execution model. In addition, the adversary may also make queries to the random oracle: $(\mathsf{SId}, f, \varphi, \mathcal{H}) \xrightarrow{\mathbf{hash}(m)} (\mathsf{SId}, f, \varphi, \mathcal{H}')$ where $\mathcal{H}'$ is defined as follows. If there exists n such that $(m, n) \in \mathcal{H}$, then $\mathcal{H}' = \mathcal{H}$ and we define $h = n$. Else a hash value h is generated at random for m and $\mathcal{H}' = \mathcal{H} \cup \{(m, h)\}$. In any case, h is returned to the adversary. A finite sequence of concrete global states is called a *concrete execution trace* if it starts with the initial global state. Obviously, since the adversary is a ppt Turing machine the length of the trace is bounded by a polynomial in the security parameter η. Also, the sequence of random coins R_Π used in the execution by the honest agents and the random oracle as well as the sequence of random coins $R_\mathcal{A}$ used by the adversary can be bounded in length by polynomials $g_\mathcal{A}(\eta)$ and $p_\mathcal{A}(\eta)$, respectively. Clearly, if R_Π and $R_\mathcal{A}$ are fixed, we obtain a uniquely determined concrete trace, which we denote by $\mathsf{Exec}_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta)$.

3 Symbolic and Computational Secrecy Properties

In this section we recall the computational definition of secrecy and introduce our new symbolic definition for secrecy.

COMPUTATIONAL SECRECY. Computational secrecy requires that no partial information is leaked to the adversary. The typical way to formalize this idea is to require that the secret s is indistinguishable from an unrelated random bitstring s' chosen (from an appropriate distribution). The secrecy of nonce variable X_{A_i} (the nonce generated by A_i in the ith role of the protocol) in protocol Π is defined as follows.

Definition 1. *Consider the experiment* $\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta)$ *parametrized by a bit* b *and that involves an adversary* $\mathcal{A}$ *against protocol* Π*. The experiment takes as input a security parameter* η *and starts by generating two random nonces* n_0 *and* n_1 *in* $C^\eta.n$*. Then the adversary* $\mathcal{A}$ *starts interacting with the protocol* Π *as in the execution described by* $\mathsf{Exec}_{\Pi,\mathcal{A}}(\eta)$*. At some point in the execution the adversary initiates a session* s *in which the role of* A_i *is executed, and declares this session under attack. In this session, the variable* $X^j_{A_i}$ *is instantiated with* n_b*. The rest of the execution is exactly as in* $\mathsf{Exec}_{\Pi,\mathcal{A}}(\eta)$*. At some point the adversary requires the two nonces* n_0 *and* n_1 *and has to output a guess* d*. The bit* d *is the result of the experiment. We define the advantage of the adversary* $\mathcal{A}$ *by:*

$$\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = \Pr\left[\mathbf{Exp}^{\text{sec\_1}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right] - \Pr\left[\mathbf{Exp}^{\text{sec\_0}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right]$$

We say that nonce $X^j_{A_i}$ *is computationally secret in protocol* Π*, and we write* $\Pi \models^{\mathsf{c}} \mathsf{SecNonce}(i,j)$ *if for every p.p.t. adversary* $\mathcal{A}$ *its advantage is negligible.*

SYMBOLIC SECRECY. As explained in the introduction, weak secrecy is not sufficient to capture the standard indistinguishability-based notion used in computational settings. The new notion of secrecy we propose here relies on the intuitively appealing concept of patterns [1]. Roughly, the pattern of an expression is obtained by replacing with $\Box$, all the subterms of the expression that are secret. In our case, a subterm T of T' is secret if, even when given T the adversary cannot verify that T has been used to construct T'. Formally, we add T to the knowledge set ϕ in the deduction relation. The ideas behind our definition of patterns are related to offline guessing attacks, where the adversary is given the weak secret and should be unable to test whether the given weak secret is indeed the one used in the observed messages.

Definition 2 (Patterns). *Given a set of closed terms* $\phi = \{M_1, M_2, \ldots, M_k\}$ *and a term* T*, we define* $\mathsf{Pat}_T(\phi) = \{\mathsf{Pat}_T^\phi(M_1), \mathsf{Pat}_T^\phi(M_2), \ldots, \mathsf{Pat}_T^\phi(M_k)\}$*, where* $\mathsf{Pat}_T^\phi(M)$ *defined recursively by:*

$$\begin{aligned}
\mathsf{Pat}_T^\phi(a) &= \begin{cases} a & \textit{if } \phi, T \vdash a \\ \square & \textit{otherwise} \end{cases} \\
\mathsf{Pat}_T^\phi(\langle M_1, M_2\rangle) &= \langle \mathsf{Pat}_T^\phi(M_1), \mathsf{Pat}_T^\phi(M_2)\rangle \\
\mathsf{Pat}_T^\phi(\{M\}^r_{\mathsf{ek}(a)}) &= \begin{cases} \{\mathsf{Pat}_T^\phi(M)\}^r_{\mathsf{ek}(a)} & \textit{if } \phi, T \vdash \mathsf{dk}(a) \textit{ or if } r \in \mathsf{Rand}_{adv} \\ \square & \textit{otherwise} \end{cases} \\
\mathsf{Pat}_T^\phi(h(M)) &= \begin{cases} h(\mathsf{Pat}_T^\phi(M)) & \textit{if } \phi, T \vdash M \\ \square & \textit{otherwise} \end{cases}
\end{aligned}$$

Pat_T^ϕ *is extended to set of messages as expected:* $\mathsf{Pat}_T^\phi(S) = \bigcup_{t \in S} \mathsf{Pat}_T^\phi(t)$.

The messages of ϕ may contain some subterms of the form $\{M\}^r_{\mathsf{ek}(a)}$ where $r \in \mathsf{Rand}_{adv}$. Because of the random coins such messages must have been build by the adversary and M should be deducible. Thus we consider ϕ augmented with such messages: $\overline{\phi} = \phi \cup \{M \mid \{M\}^r_{\mathsf{ek}(a)} \text{ subterm of } \phi\}$. For any valid message set ϕ (that is $\phi \in \mathsf{Msg}^s(\Pi)$ for some protocol Π), we can show that $\phi \vdash M$ for every $M \in \overline{\phi}$.

Definition 3 (Nonce secrecy). *Let* Π *be a protocol and* $X^j_{A_i}$ *a nonce variable occurring in some role* A_i*. We say that* $X^j_{A_i}$ *is secret in* Π *and we write* $\Pi \models^s \mathsf{SecNonce}(i, j)$*, if for every valid set of messages* $\phi \in \mathsf{Msg}^s(\Pi)$ *it holds that for every session number* s*, the symbolic nonce* $n^{a_i, j, s}$ *does not occur in* $\mathsf{Pat}_{n^{a_i, j, s}}(\overline{\phi})$.

To better appreciate these definitions, consider the following examples.

1. Let $\phi_1 = \{h(\langle n_b, n'\rangle)\} = \overline{\phi_1}$. Then $\mathsf{Pat}_{n_b}(\overline{\phi_1}) = \{\square\}$. ϕ_1 preserves the indistinguishability of n_b since, intuitively, n_b is hidden by the secret nonce n'.
2. Let $\phi_2 = \{h(\langle n_b, \{n'\}^r_{\mathsf{ek}(a)}\rangle), n'\}$ where $r \notin \mathsf{Rand}_{adv}$. Then $\overline{\phi_2} = \phi_2$ and $\mathsf{Pat}_{n_b}(\overline{\phi_2}) = \{\square, n'\}$. In this example, the encryption of n' does hide n_b.
3. Let $\phi_3 = \{h(\langle n_b, \{n'\}^r_{\mathsf{ek}(a)}\rangle)\}$ where $r \in \mathsf{Rand}_{adv}$. Then $\overline{\phi_3} = \phi_3 \cup \{n'\}$ and $\mathsf{Pat}_{n_b}(\overline{\phi_3}) = \{h(\langle n_b, \{n'\}^r_{\mathsf{ek}(a)}\rangle), n'\}$. We have that n_b occurs in $\mathsf{Pat}_{n_b}(\overline{\phi_3})$. This corresponds indeed to an attack. As n' has been encrypted by the adversary himself he knows the ciphertext. Given n_0 and n_1 he computes both $h(\langle n_0, \{n'\}^r_{\mathsf{ek}(a)}\rangle)$ and $h(\langle n_1, \{n'\}^r_{\mathsf{ek}(a)}\rangle)$ and compares them to $h(\langle n_b, \{n'\}^r_{\mathsf{ek}(a)})$ yielding the attack.
4. Let $\phi_4 = \{\{\langle h(n_b), h(n')\rangle\}^r_{\mathsf{ek}(a)}, \mathsf{dk}(a)\}$ where $r \notin \mathsf{Rand}_{adv}$. Then $\overline{\phi_4} = \phi_4$ and $\mathsf{Pat}_{n_b}(\overline{\phi_4}) = \{\{\langle h(n_b), \square\rangle\}^r_{\mathsf{ek}(a)}, \mathsf{dk}(a)\}$. Again, n_b does occur in $\mathsf{Pat}_{n_b}(\phi_1)$. For this attack an intruder may get $h(n_b)$ by decrypting and projecting the message $\{\langle h(n_b), h(n')\rangle\}^r_{\mathsf{ek}(a)}$ and compare $h(n_b)$ with $h(n_0)$ and $h(n_1)$ that he may compute from n_0 and n_1.

Our notion of secrecy has a useful equivalent formulation described in the following lemma. Informally, the lemma states that all unencrypted occurrences of the secret nonce in a set of messages are such that they occur in a term t that is hashed, and such that t itself can not be computed from ϕ and n.

Lemma 1. *Let ϕ be an arbitrary set of messages and n a nonce symbol that occurs in ϕ. n does not occur in $\mathsf{Pat}_n(\overline{\phi})$ if and only if $\overline{\phi} \not\vdash n$ and $\forall M$ subterm of ϕ such that $\overline{\phi} \vdash M$, $\forall p$ such that $M|_p = n$, so that there is no encryption along p, $\exists p' < p$ such that 1) $M|_{p'} = h(M')$ and 2) $\phi, n \not\vdash M'$.*

4 Symbolic Secrecy Is Equivalent to Computational Secrecy

To prove the soundness and the completeness of our secrecy criterion, we proceed in two steps: i) relate symbolic and concrete traces and ii) prove equivalence of the symbolic and computational notions.

RELATING SYMBOLIC AND CONCRETE TRACES. The first step linking security properties in symbolic and concrete models is to exhibit a relation between individual execution traces. The relation is similar to that developed in previous works [14,10], but our definitions and results have to deal with the use of random oracles in computational executions. In line with common practice in symbolic models, hash applications (explicitly captured as queries to the random oracle by concrete traces) are not reflected by the symbolic traces. Therefore, we define the *hash-query free* trace $\mathsf{clean\_hash}(t^c)$ associated to the concrete trace $t^c = (\mathsf{SId}^c_1, g_1, \varphi_1, \mathcal{H}_1), \ldots, (\mathsf{SId}^c_n, g_n, \varphi_n, \mathcal{H}_n)$. The trace $\mathsf{clean\_hash}(t^c)$ is the concrete trace $(\mathsf{SId}^c_{i_1}, g_{i_1}, \varphi_{i_1}, \mathcal{H}_{i_1}), \ldots, (\mathsf{SId}^c_{i_k}, g_{i_k}, \varphi_{i_k}, \mathcal{H}_{i_k})$, obtained by removing from t^c the states that are the result of a hash request.

Definition 4. *Let $t^s = (\mathsf{SId}^s_1, f_1, \phi_1), \ldots, (\mathsf{SId}^s_n, f_n, \phi_n)$ be a symbolic execution trace and let $\mathsf{clean\_hash}(t^c) = (\mathsf{SId}^c_1, g_1, \varphi_1, \mathcal{H}_1), \ldots, (\mathsf{SId}^c_n, g_n, \varphi_n, \mathcal{H}_n)$ be the hash-query free trace of concrete execution trace t_c.*

- *We say that trace t^c is a* concrete instantiation of t^s with (partial) mapping $c : \mathsf{M} \to \mathcal{C}^\eta$ *and we write $t^s \preceq^{\mathsf{c}} t^c$ if for every ℓ $(1 \leq \ell \leq n)$ it holds that $\mathsf{SId}^s_\ell = \mathsf{SId}^c_\ell$ and for every $\mathsf{sid} \in \mathsf{SId}^s_\ell$ if $f_\ell(\mathsf{sid}) = (\sigma^{\mathsf{sid}}, i^{\mathsf{sid}}, p^{\mathsf{sid}}, (a_1, \ldots, a_k))$ and $g_\ell(\mathsf{sid}) = (\tau^{\mathsf{sid}}, j^{\mathsf{sid}}, q^{\mathsf{sid}}, (a_1, \ldots, a_k))$ then $\tau^{\mathsf{sid}} = c \circ \sigma^{\mathsf{sid}}$, $i^{\mathsf{sid}} = j^{\mathsf{sid}}$ and $p^{\mathsf{sid}} = q^{\mathsf{sid}}$.*
- *Trace t^c is a* concrete instantiation with Dolev-Yao hash queries *of t^s and we write $t^s \preceq t^c$ if there exists a partial, injective function $c : \mathsf{M} \to \mathcal{C}^\eta$ such that $t^s \preceq^{\mathsf{c}} t^c$ and for every $1 \leq k \leq n$, for every message m such that $(m, h) \in \mathcal{H}_k$ for some h, there exists a term M such that $c(M) = m$ and $\phi_k \vdash M$.*

Proposition 1. *Let Π be an executable protocol. If the encryption scheme $\mathcal{AE}$ is* IND-CCA *secure, and the hash functions are random oracles, then for any p.p.t. algorithm $\mathcal{A}$*

$$\Pr\left[\exists t^s \in \mathsf{Exec}^s(\Pi) \mid t^s \preceq \mathsf{Exec}^c_{\Pi(R_\Pi),\mathcal{A}(R_\mathcal{A})}(\eta)\right] \geq 1 - \nu_\mathcal{A}(\eta)$$

where the probability is over the choice $(R_\Pi, R_\mathcal{A}) \xleftarrow{\$} \{0,1\}^{p_\mathcal{A}(\eta)} \times \{0,1\}^{g_\mathcal{A}(\eta)}$ and $\nu_\mathcal{A}(\cdot)$ is some negligible function.

The proof shares many ideas with earlier work [14,10].

SYMBOLIC SECRECY IS EQUIVALENT TO COMPUTATIONAL SECRECY. The following theorem states that the symbolic secrecy criterion is necessary and sufficient for computational secrecy to hold.

Theorem 1. *Let Π be an executable protocol and let $X^j_{A_i}$ be a nonce variable occurring in some role A_i. If the encryption scheme $\mathcal{AE}$ used in the implementation of Π is* IND-CCA *secure then $\Pi \models^s \mathsf{SecNonce}(i,j)$ if and only if $\Pi \models^c \mathsf{SecNonce}(i,j)$.*

Proof. **The "if" direction.** First, we give an ideal execution of the protocols that replaces real nonces with random strings. We show that no adversary can distinguish the modified execution, which we call the "oracle execution" from the real execution.

Next, we argue that in the oracle execution, nonces that are symbolically secret are information theoretically hidden from the computational adversary. Indeed, if the symbolic secrecy property is satisfied, by Lemma 1 the nonce occurs only in some hashed terms, and the term themselves are secret (i.e., it cannot be computed efficiently). Since in the random oracle model the hash values are independent of the hashed message, the view of the adversary is independent from the value of the secret nonces.

STEP I. We now describe the "oracle execution". Whenever the protocol dictates that an honest party encrypts some bitstring m, the party encrypts instead a randomly selected bitstring r_m of equal length. The execution keeps a table with all association (m, r_m), which we call the random associations table (RAT). The RAT is not made available to the adversary, but only to honest parties. Specifically, whenever an honest party receives encrypted messages, the party performs the appropriate decryption and recovers some plaintext. If the plaintext is some m' such that (m, m') occurs in RAT, the party treats the encryption as an encryption of m and continues its execution as normal. Otherwise, the underlying plaintext is set to m'.

Intuitively, if any adversary behaves differently in the two executions, it is because he can see the difference between encryptions of true, and random ciphertexts. Formally, if we let $\mathsf{Exec}_{\mathcal{A},\Pi}(\eta)$ be the output of adversary $\mathcal{A}$ when executed with protocol Π for security parameter η, and $\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta)$ the output of the adversary in the associated oracle execution, we have the following lemma.

Lemma 2. *Let Π be an executable protocol, and $\mathcal{A}$ an arbitrary ppt adversary. Then, if the encryption scheme $\mathcal{AE}$ used in the implementation of Π is* IND-CCA *secure, then* $\Pr\left[\mathsf{Exec}_{\mathcal{A},\Pi}(\eta) = 1\right] - \Pr\left[\mathsf{Exec}^o_{\mathcal{A},\Pi}(\eta) = 1\right]$ *is negligible.*

Notice that we can apply the above lemma for the case when the execution that is considered is used in the experiment $\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}_{\mathcal{A},\Pi}}(i,j)(\eta)$, for some b, i, j. If we write $\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}^o_{\mathcal{A},\Pi}}(i,j)(\eta)$ for the corresponding oracle execution, we obtain that there exists some negligible function $\nu_{i,j,b}$ such that

$$\Pr\left[\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = 1\right] - \Pr\left[\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta) = 1\right] = \nu_{i,j,b}(\eta) \qquad (1)$$

STEP II. Next, we associate symbolic traces to the computational traces of the oracle execution. This enables us to reason about an adversary's success in the oracle execution (which is conceptually simpler). The association is in fact the one in the proof of Proposition 1, with an additional parsing step necessary to take into account the random association table that we detail below. In addition to access to the keys and the randomness of the parties, the parsing procedure also uses access to the random association table, and is as follows: the first step in processing some message m' is a search in the random association table. If (m, m') occurs in the RAT, then the procedure proceeds as before, with m' replaced by m, otherwise the procedure remains unchanged.

Next, we argue that the symbolic traces obtained as above are valid execution traces, and moreover, that they are included among the traces of the execution of Π. The formalization is given in the next lemma.

Lemma 3. *The symbolic traces of* $\mathsf{Exec}^o(\Pi, \mathcal{A})$ *are valid with overwhelming probability and* $\mathsf{Exec}^o_{\mathcal{A},\Pi} \subseteq \mathsf{Exec}_{\mathcal{A},\Pi}$.

STEP III. Finally, we prove that if $\mathcal{AE}$ is IND-CCA secure then $\Pi \models \mathsf{SecNonce}^s(i,j) \Rightarrow \Pi \models^{\mathsf{c}} \mathsf{SecNonce}(i,j)$. For an arbitrary adversary $\mathcal{A}$ against the secrecy of nonce $X^j_{A_i}$ recall that we write $\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(\eta)$ for the oracle version of the experiment defining secrecy of nonce $X^j_{A_i}$. Let $\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\mathcal{A},\Pi}}(\eta)$ be the corresponding advantage functions. By definition we have that:

$$\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = \Pr\left[\mathbf{Exp}^{\text{sec\_1}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right] - \Pr\left[\mathbf{Exp}^{\text{sec\_0}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right]$$

$$\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta) = \Pr\left[\mathbf{Exp}^{\text{sec\_1}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta){=}1\right] - \Pr\left[\mathbf{Exp}^{\text{sec\_0}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta){=}f1\right]$$

By subtracting, using Equation 1, we obtain that for some negligible function ν

$$\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}_{\Pi,\mathcal{A}}}(i,j)(\eta) = \mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta) + \nu(\eta) \tag{2}$$

Finally, we show that in the oracle execution the advantage $\mathbf{Adv}^{\text{sec}}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(i,j)(\eta)$ of any adversary $\mathcal{A}$ is negligible since nonces that are symbolically secret are informational theoretically hidden from the adversary. This can be seen as follows.

Consider the symbolic trace ϕ that corresponds to the execution of the experiment $\mathbf{Exp}^{\text{sec\_}b}_{\mathsf{Exec}^o_{\Pi,\mathcal{A}}}(\eta)$, up to the point when the adversary is given the nonces and he is asked to determine the bit b. Let s be the id of the session under attack, and let $n^{a,j,s}$ be the symbolic nonce that corresponds to the nonce under attack. By Lemma 3, the trace ϕ is with overwhelming probability a Dolev-Yao trace of protocol Π. By the hypothesis of the theorem $\Pi \models^s \mathsf{SecNonce}(i,j)$ and therefore by Lemma 1, all occurrences of $n^{a,j,s}$ in ϕ that are not under an honest encryption are in some term T_i that appears under a hash, and T_i is nondeductible from $\phi, n^{i,j,s}$. Let t_i be the bitstrings that correspond to the terms T_i. We conclude by observing that in the real execution, the adversary may observe the values $c_1 = h(t_1), c_2 = h(t_2), \ldots$, but provided that it does not query $t_1, t_2, \ldots$ to the random oracle, their values (and thus in particular the value of the secret nonce) are independent from the $c_1, c_2, \ldots$. Since all queries to the random oracle are the images of deductible terms, we conclude that $\mathcal{A}$ does not request $h(t_i)$, for all i.

The "only if" direction. It is important to observe that if a message M is deducible from a set of messages $M_1, M_2, \ldots, M_n$, the associated deduction tree τ can be translated into an (efficient) program $\overline{\tau}$ which given the bit-string representations of m_i for M_i $(i = 1, 2, \ldots, n)$ computes the bit-string representation m of M.

We proceed as follows. Assume that for some symbolic trace ϕ, the symbolic nonce $n^{a_i,j,s}$ occurs in $\mathsf{Pat}_{n^{a_i,j,s}}(\overline{\phi})$, starting from Lemma 1 we can show that there exist a term $M \in \overline{\phi}$ and a deduction tree τ such that: 1) $\tau(\phi, n^{a_i,j,s})$ yields message M and 2) for $n \neq n^{a_i,j,s}$, $\tau(\phi, n)$ does not yield M. Since $M \in \overline{\phi}$, we know that there also exists a deduction tree π such that $\pi(\phi)$ yields M.

Based on the above, we construct a two-stage adversary against secrecy of nonce $X^j_{A_i}$. In the first stage, the adversary produces a computational representation ϕ^{c} of the trace ϕ (by simply following the instructions of the Dolev-Yao adversary that defines ϕ). Once ϕ is created, it requests the two values of the nonce $n^{a_i,j,s}$ and receives from the experiment n_b and n_{1-b}. Then it computes $m_b = \tau(\phi^{\mathsf{c}}, n_b)$ for $b = 0, 1$ and $m = \pi(\phi^{\mathsf{c}})$, and retrieves b by comparing m with m_0 and m_1.

5 Decidability of Symbolic Secrecy

In this section, we show that our notion of secrecy is decidable. We present an NP-procedure that decides nonce non-secrecy for the case of a bounded number of sessions (that is, adversaries are allowed only a fixed number of $\mathbf{new}$ queries)[2].

Without loss of generality, we assume that all of the $\mathbf{new}$ queries are performed at the beginning of the execution. Our decision procedure starts by guessing the sequence of these requests together with the identities of the agents involved. Then, the procedure guesses an interleaving for the execution. Using standard techniques [15], such executions can be translated to constraint systems. We recall their definition:

Definition 5. *A* constraint system *C is a finite set of expressions $T_i \Vdash \mathit{tt}$ or $T_i \Vdash u_i$, where T_i is a non empty set of terms, tt is a special symbol that represents an always deducible term, and (for $1 \leq i \leq n$) u_i is a term such that:*

- *$T_i \subseteq T_{i+1}$, for all $1 \leq i \leq n - 1$;*
- *if $x \in \mathrm{var}(T_i)$ then $\exists j < i$ such that $T_j = \min\{T \mid T \Vdash u \in C, x \in \mathrm{var}(u)\}$ (for the inclusion relation) and $T_j \subsetneq T_i$.*

The left-hand side *(*right-hand side*) of a constraint $T \Vdash u$ is T (respectively u). The* left-hand side *of a constraint system C, (for which we write $\mathit{lhs}(C)$), is the maximal set of messages T_n. By $\perp$ we denote the unsatisfiable system.*

The left-hand side of a constraint represents the messages already sent on the network, while the right-hand side represents the message expected by an agent in order to perform the next protocol step. A *solution* of a constraint system C is a ground substitution σ such that $T\sigma \vdash_{\mathsf{Rand}_{adv}} u\sigma$ for any $T \Vdash u \in C$. We say that *C preserves nonce secrecy* of n if there does not exist a solution σ of C such that n occurs in $\mathsf{Pat}_n(\overline{\mathit{lhs}(C)\sigma})$.

The transformation of protocols into constraint systems yields systems that are well-formed. A constraint system E is *well-formed* if 1) any subterm of E of the form $\mathsf{dk}(t')$ is such that t' is an agent identity and 2) any subterm of E of the form $\{t_1\}^r_{t_2}$ is such that $r \in \mathsf{Rand}$ and $r \notin \mathsf{Rand}_{adv}$. The following theorem states that our notion of nonce secrecy (Section 3) is decidable for a bounded number of sessions.

Theorem 2. *The following problem is co-NP complete:*
Given: *A well-formed constraint system C and a nonce n.*
Decide: *Does C preserve the nonce secrecy of n?*

[2] As for the standard deducibility-based notions, nonce secrecy is undecidable for an unbounded number of sessions.

The decision procedure for nonce secrecy preservation works as follows. First, given an arbitrary constraint system we reduce it to a *solved* system using non-deterministic transformation rules similar to those in [8]. A constraint system is *solved* if it is different from $\perp$ and each of its constraints are of the form $T \Vdash tt$ or $T \Vdash x$ where x is a variable. Second, we check whether n occurs in $\mathsf{Pat}_n(\overline{lhs(C)})$. If not, we check whether C can further be simplified into a solved form that does not preserve nonce secrecy, and so on. Note that although for standard deducibility-based notions decision procedures can stop as soon as the constraint system has been transformed into solved form, for our secrecy notion further transformations might be necessary. NP-hardness is proved analogously to the case of standard deducibility-based notions [16].

References

1. M. Abadi and P. Rogaway. Reconciling two views of cryptography. In *IFIP TCS 2000*, volume 1872 of *LNCS*, pages 3–22, August 2000.
2. A. Armando, D. Basin, Y. Boichut, Y. Chevalier, L. Compagna, J. Cuellar, P. H. Drielsma, P. Heám, O. Kouchnarenko, J. Mantovani, S. Mödersheim, D. von Oheimb, M. Rusinowitch, J. Santiago, M. Turuani, L. Viganò, and L. Vigneron. The Avispa tool for the automated validation of internet security protocols and applications. In *CAV 2005*, volume 3576 of *LNCS*, 2005.
3. M. Backes and I. Christian Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *STACS 2003*, pages 675–686, 2003.
4. M. Backes and B. Pfitzmann. Relating cryptographic und symbolic key secrecy. In *Proc. 26th IEEE Symposium on Security and Privacy (SSP'05)*, pages 171–182, 2005.
5. M. Bellare and P. Rogaway. Entity authentication and key distribution. In *Crypto'93*, volume 773 of *LNCS*, pages 232–249, 1993.
6. B. Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Proc. 14th IEEE Computer Security Foundations Workshop (CSFW'01)*, pages 82–96, 2001.
7. R. Canetti and J. Herzog. Soundness of formal encryption in the presence of active adversaries. In *TCC 2006)*, LNCS, 2006. To appear.
8. H. Comon-Lundh and V. Shmatikov. Intruder deductions, constraint solving and insecurity decision in presence of exclusive or. In *LICS 2003*, pages 271–280, 2003.
9. V. Cortier, S. Kremer, R. Küsters, and B. Warinschi. Computationally sound symbolic secrecy in the presence of hash functions. Research Report 2006/218, Cryptology ePrint Archive, June 2006. 31 pages.
10. V. Cortier and B. Warinschi. Computationally Sound, Automated Proofs for Security Protocols. In *ESOP 2005*, volume 3444 of *LNCS*, pages 157–171, 2005.
11. S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28:270–299, 1984.
12. P. Gupta and V. Shmatikov. Towards computationally sound symbolic analysis of key exchange protocols. In *FMSE 2005*, pages 23–32, 2005.
13. R. Janvier, Y. Lakhnech, and L. Mazaré. Computational soundness of symbolic analysis for protocols using hash functions. In *ICS'06*, 2006. To appear.
14. D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *TCC 2004*, volume 2951 of *LNCS*, pages 133–151, 2004.
15. J. K. Millen and V. Shmatikov. Constraint solving for bounded-process cryptographic protocol analysis. In *CCS 2001*, pages 166–175, 2001.
16. M. Rusinowitch and M. Turuani. Protocol Insecurity with Finite Number of Sessions and Composed Keys is NP-complete. *Theoretical Computer Science*, 299:451–475, April 2003.

Some Results on Average-Case Hardness Within the Polynomial Hierarchy

A. Pavan[1,⋆], Rahul Santhanam[2], and N.V. Vinodchandran[3,⋆⋆]

[1] Department of Computer Science, Iowa State University
[2] Department of Computer Science, Simon Fraser University
[3] Department of Computer Science and Engineering, University of Nebraska-Lincon

Abstract. We prove several results about the average-case complexity of problems in the Polynomial Hierarchy (PH). We give a connection among average-case, worst-case, and non-uniform complexity of optimization problems. Specifically, we show that if P^{NP} is hard in the worst-case then it is either hard on the average (in the sense of Levin) or it is non-uniformly hard (i.e. it does not have small circuits).

Recently, Gutfreund, Shaltiel and Ta-Shma (*IEEE Conference on Computational Complexity, 2005*) showed an interesting worst-case to average-case connection for languages in NP, under a notion of average-case hardness defined using uniform adversaries. We show that extending their connection to hardness against quasi-polynomial time would imply that NEXP doesn't have polynomial-size circuits.

Finally we prove an unconditional average-case hardness result. We show that for each k, there is an explicit language in P^{Σ_2} which is hard on average for circuits of size n^k.

1 Introduction

Average-case complexity is one of the central concepts in complexity theory. There are several different reasons for studying average-case complexity. The notion of being hard on average is fundamental to cryptography [Gol01, Gol04], since the security of most cryptographic protocols is conditioned on the assumption that certain problems such as factoring and discrete logarithm problem are hard on average. Notions of average-case complexity also appear naturally in the theory of pseudo-randomness [BM84, Yao82], learning theory [JS05] and the study of heuristics for solving NP-complete problems [ART06].

Does the existence of a worst-case hard problem (say, with respect to polynomial-size circuits) in a complexity class $\mathcal{C}$ imply the existence of an average-case hard problem in the class? This question is particularly significant for the case of NP in part because of its connections to cryptography and the theory of approximation. Answering the question positively for NP would enable us to base cryptography on

[⋆] Research supported in part by NSF grant CCF-0430807, and a Big12 Faculty Fellowship.

[⋆⋆] Research supported inpart by NSF grant CCF-0430991.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 188–199, 2006.

NP-hardness rather than on the hardness of specific algebraic/number-theoretic problems such as discrete logarithm and factoring. From the perspective of hardness of approximation, there is a recent line of work [Fei02, Ale03] showing that average-case hardness of NP problems would imply much better inapproximability results for certain natural problems in NP than that are currently known.

For "large enough" complexity classes such as EXP and PSPACE, it is known that worst-case hardness implies average-case hardness. This follows from generic hardness amplification techniques [STV01, TV02] which were developed in the context of the theory of pseudo-randomness. However, for NP or other classes in the polynomial-time hierarchy (PH), the techniques that work for EXP or PSPACE are known to fail [BT03, Vio05], and the question of whether worst-case hardness implies average-case hardness for NP and PH remains unsolved.

Our Results

We consider various notions of average-case hardness that have been defined in the literature, and investigate the possibility of constructing languages within the polynomial-time hierarchy that are average-case hard according to these notions.

First, we consider Levin's framework for average-case complexity [Lev86]. Informally, in this framework, a problem L is easy on average if for every polynomial-time samplable distribution μ, there is some algorithm which solves L and halts in polynomial time with high probability over the distribution μ. It is a longstanding open problem whether the assumption that NP is easy on average under this notion implies NP = P. It is also not known if there is an oracle under which the implication would not follow (and hence would require a non-relativizing technique to prove, assuming it is true). The analogous question for Σ_k^{P}, for any $k > 1$ also remains open.

We ask a weaker question: is there any non-trivial easiness assumption about PH which in conjunction with the assumption about easiness on average imply that NP = P? A natural assumption to consider is the assumption of *non-uniform* easiness, i.e., solvability by polynomial-size circuits. Our first theorem is along this direction.

Theorem 1.1. *If* $\mathrm{NP} \neq \mathrm{P}$*, then* $\mathrm{NP} \not\subseteq \mathrm{P/poly}$ *or* P^{NP} *is average-case hard.*

An immediate corollary of this result is that if $\mathrm{P}^{\mathrm{NP}} \neq \mathrm{P}$, then either P^{NP} is non-uniformly hard or P^{NP} is average-case hard. P^{NP} has a natural interpretation as the class of optimization problems whose decision versions are in NP, thus we get that for optimization problems, worst-case hardness implies either average-case hardness or non-uniform hardness.

The nonuniform hardness in the above theorem refers to worst-case nonuniform hardness. We consider the possibility of improving this to average-case nonuniform hardness. As our second main result, we obtain the following improvement to the above theorem (for a more precise statement see Section 3).

Theorem 1.2. *If* NP $\neq$ P*, then either* P^{NP} *is average-case hard, or there is a language* L *in* NP *such that for every* k *there is a polynomial-time samplable distribution* μ *and* L *is average-case hard for* n^k*-size circuits with respect to distribution* μ*.*

A more restrictive notion (than the one due to Levin) of average-case easiness that has gained prominence recently is the notion of easiness against uniform adversaries. A language L is said to be easy against a class $\mathcal{C}$ of adversaries if there is a fixed polynomial-time simulation of L such that no adversary in $\mathcal{C}$ outputs an instance on which the simulation differs from L. Note that in this setting, the simulation is independent of the adversary. This is more restrictive than Levin's notion, since in Levin's setting the running time of the simulation can depend on the running time of the adversary. A recent work of Gutfreund, Shaltiel and Ta-Shma [GSTS05] shows that under this notion the average-case complexity of NP is same as its worst-case complexity. In particular they show that if every language in NP is easy against polynomial-time adversaries then in fact NP = P. Their result has been further refined by Atserias [Ats06].

The question we ask is: how relevant is the technique of [GSTS05] for proving an average-case to worst-case connection for NP in Levin's framework? In particular if we allow the simulation to run for more than polynomial time (say quasi-polynomial time) can we extend the results of Gutfreund, Shaltiel and Ta-Shma to get an average-case to worst-case equivalence for NP? We show that such a result would imply a groundbreaking circuit lower bound result, and hence is unlikely to be provable using current techniques.

Theorem 1.3. *If* $\mathrm{NP} \subseteq \mathit{quasi}_{\mathrm{P}}\!-\!\mathsf{QP}$ *implies* $\mathrm{NP} \subseteq \mathsf{QP}$*, then* $\mathrm{NEXP} \not\subseteq \mathrm{P/poly}$.

We refer the reader to Section 2 for the definition of "$\mathit{quasi}_{\mathrm{P}}$", which formalizes the notion of easiness used in [GSTS05].

Next we consider the question of whether known worst-case lower bounds in PH can be extended to average-case lower bounds. To be specific, we ask: which level in the polynomial-time hierarchy has languages that are average-case hard for circuits of size n^k? Kannan [Kan82] showed using a nonconstructive argument that for any fixed k, there is a language in the second level of PH (more precisely $\Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$) that cannot not be computed by circuits of size n^k. Since then there have been a series of attempts to prove better upper bounds on the complexity of such a language. The current best upper bound known [Cai01] is $\mathrm{S}_2^{\mathrm{P}}$, which is a subclass of $\Sigma_2^{\mathrm{P}} \cap \Pi_2^{\mathrm{P}}$. There has been related work on constructing explicit languages in low levels of PH that do not have circuits of size n^k (the $\mathrm{S}_2^{\mathrm{P}}$ upper bound is proved non-constructively). Miltersen, Vinodchandran and Watanabe [MVW99] showed a constructive upper bound of $\mathrm{P}^{\Sigma_2^{\mathrm{P}}}$; Cai and Watanabe [CW03] improved the upper bound to Σ_2.

Note that since an average-case to worst-case connection is not known within PH, we cannot directly use the results above to show an average-case hardness result. Nevertheless, we strengthen the technique of Miltersen, Vinodchandran and Watanabe to show that for each k, there is an explicit language in $\mathrm{P}^{\Sigma_2^{\mathrm{P}}}$

which cannot be *approximated* on significantly more than 1/2 the inputs of any input length by circuits of size n^k (refer to Section 2 for the precise definition of (n^k, n^k)-hard in the following).

Theorem 1.4. *For each k, there is a language L_k in $\mathrm{P}^{\Sigma_2^{\mathrm{P}}}$ such that L_k is (n^k, n^k)-hard.*

2 Preliminaries

We say that $\mu = (\mu_1, \mu_2, \cdots)$ is an *ensemble of distributions* if each μ_n is a distribution over Σ^n. We often use the word distribution instead of ensemble of distributions. A distribution μ is p-samplable if there is a probabilistic algorithm A such that for every $x \in \Sigma^n$

$$Pr[A(1^n) = x] = \mu_n(x).$$

We use Levin's notion of *average polynomial-time* [Lev86]. In his definition of average-polynomial time, Levin considered a distribution over Σ^* rather than an ensemble of distributions. However, many times it is more convenient to consider an ensemble of distributions rather than a single distribution. Gurevich [Gur91] and Impagliazzo [Imp95] showed that Levin's definition can be adapted to the case of an ensemble of distributions. We follow this adaptation.

Let $\mu = (\mu_1, \mu_2, \cdots)$ be an ensemble of distributions. We associate distribution μ_{assoc} over Σ^*, to the ensemble $(\mu_1, \mu_2, \cdots)$, as follows: if x is a string of length n, then

$$\mu_{assoc}(x) = \frac{6}{\pi^2}\frac{1}{n^2}\mu_n(x).$$

Definition 2.1. *([Lev86]) Let L be a language and $\mu = (\mu_1, \mu_2, \cdots)$ be a distribution. We say (L, μ) is in* Average-P *if there is a machine M that decides L and a constant $k \geqslant 1$,*

$$\sum_x \frac{(T_M(x))^{1/k}}{|x|}\mu_{assoc}(x) < \infty.$$

Remark. In Levin's notion of Average polynomial-time, a single distribution over Σ^* is used instead of an ensemble of distributions as above.
We find the following observation to be useful.

Observation 2.2. *Let μ be an ensemble of distributions, and let (L, μ) in* Average-P. *There exists a Turing machine M that accepts L such that for every polynomial $p(.)$, there exists a constant $l > 0$, and for all but finitely many n,*

$$\Pr_{x \in \mu_n}[M(x) \textit{ does not halt in } n^l \textit{ steps}] < 1/p(n).$$

Given a complexity class $\mathcal{C}$, let Dist$\mathcal{C}$ denote the class of distributional problems (L, μ), where $L \in \mathcal{C}$ and μ is a p-samplable ensemble. Now whether Dist$\mathcal{C} \subseteq$ Average-P is the average-case analogue of whether $\mathcal{C} \subseteq$ P. Given a class $\mathcal{C}$, we say that $\mathcal{C}$ *is easy on average* if Dist$\mathcal{C} \subseteq$ Average-P.

We can adapt Levin's notion of average polynomial time to function classes also. The following observation is easy to prove.

Observation 2.3. *If* P^{NP} *is easy on average, then* $\mathsf{PF}^{\mathrm{NP}}$ *is easy on average.*

We also consider the notions of average-case complexity under the uniform distribution in nonuniform models of computation.

Definition 2.4. *Let s and h be functions from $\mathbb{N}$ to $\mathbb{N}$. A language L is called (s, h)-hard if for every $s(n)$-size circuit family $C = (C_1, C_2, \cdots)$*

$$\Pr_{x \in \Sigma^n}[L(x) = C_n(x)] \leqslant 1/2 + 1/h(n).$$

Here x is drawn uniformly at random from Σ^n.

Definition 2.5. *A distributional problem (L, μ) is in $\mathsf{HSIZE}(n^k)$, if for every polynomial p, there is a n^k-size circuit family $C = (C_1, C_2, \cdots)$ such that for all but finitely many n*

$$\Pr_{x \in \mu_n}[L(x) \neq C_n(x)] \leqslant 1/p(n).$$

In this paper, we also study a notion of easy on average that is different from Levin's notion of easy on average. This notion naturally arises in the theory of pseudo-randomness and uniform derandomization. This notion was implicit in the work of Impagliazzo and Wigderson [IW98]. Kabanets [Kab01] made this explicit and defined "pseudo classes."

Definition 2.6. *([Kab01]) Let $\mathcal{C}$ be a complexity class. A language L is in pseudo$_{\mathrm{P}}-\mathcal{C}$ if there is a language L' in $\mathcal{C}$ such that for every polynomial-time machine R for all but finitely many n, $R(1^n) \notin L \Delta L'$.*

Thus if a language L is in *pseudo*$_{\mathrm{P}}-\mathcal{C}$, there is a simulation L' for L and no adversary R can find places where L' and L differ. We obtain the class *quasi*$_{\mathrm{P}}-$P (defined in [vMS05]) by allowing the adversary R to output more than one string.

Definition 2.7. *A language L is in quasi$_{\mathrm{P}}-\mathcal{C}$, if there is a language L' in $\mathcal{C}$ such that for every polynomial-time machine R for all but finitely many n, no output of $R(1^n)$ belongs to $L \Delta L'$.*

A consistent circuit for SAT is a circuit that can err only on one-side. More formally,

Definition 2.8. *We say a circuit C is* consistent circuit *for* SAT*, if C outputs a satisfying assignment whenever it says a formula is satisfiable.*

We use the following known results in our proofs.

Theorem 2.9. *[BCK+ 96] Assume* $\mathrm{NP} \subseteq \mathrm{P/poly}$*. There is a* $\mathrm{ZPP}^{\mathrm{NP}}$ *machine* M *such that* M *on input* 1^n *either outputs "?" or outputs a circuit for* SAT_n*. Probability that* M *outputs "?" is at most* $1/2^n$*.*

Theorem 2.10. *[FPS03] For every* k*, there is a* $\mathrm{ZPP}^{\mathrm{NP}}$ *algorithm* M *such that if* SAT *does not have* n^{k+2}*-size circuits at length* n *then* M *on input* 1^n *either outputs "?" or outputs a list of formulas* $\phi_1, \phi_2, \cdots, \phi_m$*,* $m \leqslant n^{2k}$*, such that*

- $\Pr[M(1^n) =?] \leqslant 1/2^n$
- *If* $M(1^n)$ *outputs* $\phi_1, \cdots \phi_m$*, then for every* n^k*-size consistent circuit* C*, there exists* i*,* $1 \leqslant i \leqslant m$ *such that* $C(\phi_i) \neq \mathrm{SAT}(\phi_i)$*.*

3 Easiness on Average Versus Nonuniform Easiness

In this section we show results that connect the worst-case, average-case, and non-uniform hardness of the class P^{NP}.

Theorem 1.1 *If* $\mathrm{P} \neq \mathrm{NP}$*, then at least one of the following statements is true.*

- P^{NP} *is not easy on average.*
- NP *is not in* P/poly*.*

Proof. Assume that NP is in P/poly and P^{NP} is easy on average. Since NP is in P/poly, by Theorem 2.9 there is a $\mathrm{ZPP}^{\mathrm{NP}}$ machine M that on input 1^n outputs a circuit for SAT_n with high probability. Assume that $M(1^n)$ needs n^k random bits. Define a function f as follows:

$$f(1^n, r) = M(1^n, r).$$

where $|r| = n^k$, and $M(1^n, r)$ is the output of M when it is given r as random seed. Clearly, $f \in \mathsf{PF}^{\mathrm{NP}}$. Since P^{NP} is easy on average, by Observation 2.3, for every p-samplable distribution μ, (f, μ) can be computed in polynomial-time on average. Consider the following distribution $\mu = (\mu_1, \mu_n, \cdots)$, where μ_n randomly and uniformly picks a string r of length n^k and outputs $\langle 1^n, r\rangle$.

Let N be a machine that computes f in average polynomial time with respect to μ. By Observation 2.2, there exists $l > 0$ such that

$$Pr_r[N(1^n, r) \text{ does not halt in } n^l \text{ steps}] \leqslant 1/n^2.$$

Since N computes f

$$Pr_r[N(1^n, r) =?] \leqslant 1/2^n.$$

Thus if we randomly pick r, probability that $N(1^n, r)$ either takes more than n^l time or outputs "?" is at most $1/n$. Thus if we randomly pick r and stop the computation of $N(1^n, r)$ after n^l steps, then with very high probability it outputs a circuit for SAT_n. This gives a probabilistic polynomial-time algorithm that can compute circuits for SAT.

Thus $\mathrm{SAT} \in \mathrm{BPP}$ and so $\mathrm{NP} \subseteq \mathrm{BPP}$. Buhrman, Fortnow and Pavan [BFP05] showed that if NP is easy on average, then $\mathrm{BPP} = \mathrm{P}$. Thus $\mathrm{NP} = \mathrm{P}$.

This theorem has the following interesting corollary.

Corollary 3.1. *If* P^{NP} *is hard in the worst-case, then either it is non-uniform hard or average-case hard.*

Theorem 1.1 says that if NP does not equal to P, then either P^{NP} is hard on average or there is a language in NP that is not in $\mathsf{SIZE}(n^k)$ for every $k > 1$. This language in NP is worst-case hard in the non-uniform model. Can we make this language to be average-case hard in the non-uniform model? We show the following:

Theorem 1.2 *If* $\mathrm{P} \neq \mathrm{NP}$*, then at least one of the following statements is true.*

- P^{NP} *is not easy on average.*
- *There is a language* L *in* NP *such that for every* k *there is a p-samplable distribution* μ *such that* $(L, \mu) \notin \mathsf{HSIZE}(n^k)$*.*

Remark. In this result, the distribution μ depends on the constant k. Making the distribution independent of k yields the much sought average-case to worst-case connection for P^{NP}.

The theorem follows from the following two Lemmas. We omit the proofs due to lack of space. Proof of these Lemma 3.2 makes crucial use of Theorem 2.10.

Lemma 3.2. *If* $\mathrm{P} \neq \mathrm{NP}$*, then at least one of the following statements is true.*

- P^{NP} *is not easy on average.*
- *For every* k *there is a p-samplable distribution* μ*, and infinitely many* n *such that for every* n^k*-size consistent circuit fancily* $C = (C_1, C_2, \cdots)$ *for* SAT

$$\Pr_{x \in \mu_n}[C_n(x) \neq \mathrm{SAT}(x)] \geqslant 1/n^{4k}.$$

Lemma 3.3. *Assume that the following statement holds: For every* k *there is a p-samplable distribution* μ*, and infinitely many* n *such that for every* n^k*-size consistent circuit family* $C = (C_1, C_2, \cdots)$ *for* SAT*,*

$$\Pr_{x \in \mu_n}[C_n(x) \neq \mathrm{SAT}(x)] \geqslant 1/n^{4k}.$$

Then, there is a language L *in* NP *such that for every* k*, there is a p-samplable distribution* μ *and*

$$(L, \mu) \notin \mathsf{HSIZE}(n^k).$$

4 On the Difficulty of Showing Easiness on Average Implies Easiness in the Worst Case

A recent result by Gutfreund, Shaltiel and Ta-Shma [GSTS05] on worst-case to average-case reductions for NP problems states that if there is a simulation of SAT in polynomial time which fools all polynomial-time adversaries, then $\mathrm{NP} = \mathrm{P}$.

Theorem 4.1. *If* NP $\subseteq$ $\mathit{quasi}_{\mathrm{P}}{-}\mathrm{P}$*, then* NP = P*.*

Theorem 4.1 can be interpreted as follows. If SAT is not in polynomial time, then for any polynomial-time algorithm A purporting to solve SAT, there is an adversary—a polynomial time procedure—that for each n produces a small list of candidate *counter-examples* of size n. Namely the adversary outputs a list of formulae such that there is at least one formula ϕ in the list for which $A(\phi) \neq SAT(\phi)$. In fact, the proof of Theorem 4.1 gives an upper bound of 3 on the size of the list.

It is crucial to the proof of Theorem 4.1 that the adversary has more resources than the simulating class. Indeed, the proof of Theorem 4.1 proceeds via construction of an adversary which simulates an algorithm A purporting to solve SAT. On the other hand, showing an average-case to worst-case connection for NP under Levin's notion would mean that if NP $\neq$ P, then there is a distribution μ such that (SAT, μ) is not solved on average by any polynomial-time algorithm, where the algorithm may take more time than is required to sample from μ. Thus intuitively, if the method of [GSTS05] is to be applicable to showing an average-case to worst-case connection for NP, it should be possible to extend Theorem 4.1 to a setting where the simulating class has more power than the adversary. We show that this is unlikely using current techniques (indeed, using relativizing techniques) since NEXP $\not\subseteq$ P/poly is a consequence.

We will actually show that NEXP $\neq$ MA, which implies the circuit lower bound by the following result of Impagliazzo, Kabanets and Wigderson:

Theorem 4.2. *[IKW02]* NEXP $\neq$ MA *if and only if* NEXP $\not\subseteq$ P/poly*.*

We consider two cases, the first where NP is somewhat easy in the worst case, and the second where NP is somewhat hard according to the notion of hardness in [GSTS05]. In both cases, we show that MA $\neq$ NEXP follows. Thus MA $\neq$ NEXP would follow from an average-case to worst-case connection. In the first case, we use standard techniques, and in the second case, we use the "easy witness" method of Kabanets [Kab01] and Impagliazzo, Kabanets and Wigderson [IKW02]. Let QP denote the class of languages that can be decided in deterministic quasi-Polynomial time, and NQP is the nondeterministic analogue of QP.

Lemma 4.3. *If* NP $\subseteq$ QP*, then* MA $\neq$ NEXP*.*

Proof. We will prove something even stronger, namely that MA $\neq$ EXP.

By Lautemann's theorem [Lau83], MA $\subseteq \Sigma_2^{\mathrm{P}}$. If NP $\subseteq$ QP, then MA $\subseteq \Sigma_2^{\mathrm{P}} = \mathrm{NP}^{\mathrm{NP}} \subseteq \mathrm{NP}^{\mathsf{QP}} \subseteq$ NQP.

By padding, if NP $\subseteq$ QP, then NQP = QP, and hence MA $\subseteq$ QP. By the hierarchy theorem for deterministic time, EXP $\not\subseteq$ QP, and hence MA $\neq$ EXP.

Next, we show that a superpolynomial lower bound on average-case hardness in the framework of [GSTS05] would also separate MA and NEXP. We will need the optimal construction of pseudo-random generators due to Umans [SU05, Uma02] in the proof.

Theorem 4.4. *There is a function* $G : \{0,1\}^{2^m} \times \{0,1\}^{O(m)} \to \{0,1\}^{m^s}$ *computable in polynomial time such that if* f *is a Boolean function on* m *bits which doesn't have circuits of size* m^{3s}*, then for any circuit* C *of size* m^s*,* $|\Pr_{y \in \{0,1\}^{m^s}}(C(y) = 1) - \Pr_{x \in \{0,1\}^m}(C(G(f,x)) = 1)| < 1/m^s$.

Lemma 4.5. *If* NP $\not\subseteq$ *quasi*$_{\mathrm{P}}$$-$QP*, then* MA $\neq$ NEXP.

Proof Sketch. Fix a language L in NP. We attempt to simulate L in deterministic time $2^{polylog(n)}$ on inputs of length n as follows. For an input x of length n, we interpret a witness for x as the truth table of a Boolean function (rounding the witness size upwards to a power of 2). We search for witnesses describable by small circuits, i.e., circuits of size $polylog(n)$. If we find such a witness for x, we accept x, otherwise we reject. Clearly, the search can be implemented exhaustively in time $2^{polylog(n)}$.

Since NP $\not\subseteq$ *quasi*$_{\mathrm{P}}$ $-$ QP, there is an $L \in$ NP such that the simulation above fails for L. Moreover, there is a polynomial time machine B outputting a list of instances such that the simulation fails on at least one of the instances. We will use the machine B to derive a simulation of MA in non-deterministic subexponential time with small advice, and then use a hierarchy theorem to show that this implies a separation of MA and NEXP.

We show that for any language $L' \in$ MA, $L' \in i.o.\mathrm{NTIME}(2^{O(m)})/O(m)$. The basic idea is that the machine B can be used to derandomize a Merlin-Arthur machine accepting L' infinitely often, given small advice. This is because for infinitely many input lengths n, there is at least one instance $y \in L$ of length n output by B such that none of the witnesses for y are describable by small circuits. Thus, if we knew y, we could non-deterministically compute the truth table of a hard function by merely guessing and verifying a witness for y. Once we have the truth table of a hard function, we could use Theorem 4.4 to derandomize a polynomial-time Merlin-Arthur machine and simulate it's computation in NTIME$(2^{O(m)})$, where m is the length of the input to the machine.

We do not know y but B does produce a small list containing y. Thus, given a small amount of advice telling us the index of y in the list, we can determine y. We also do not know precisely for which input lengths B produces a list containing at least instance in L with hard witnesses. But we know that this happens infinitely often, and we can again use a small amount of advice to point to the right input lengths. We omit the details in this sketch.

Now assume, for the purpose of contradiction, that MA = NEXP. Since MA $\subseteq$ EXP $\subseteq$ NEXP, we have that EXP = NEXP. This implies that there is some constant c such that NE $\subseteq$ DTIME(2^{n^c}) (since NE has a complete language, and a deterministic time upper bound for that complete language also holds for any language in NE). It follows that NE$/O(n) \subseteq$ DTIME$(2^{n^c})/O(n)$. We have that MA $\subseteq i.o.$NE$/O(n) \subseteq i.o.$DTIME$(2^{n^c})/O(n)$. Since MA = EXP by assumption, we have that EXP $\subseteq i.o.$DTIME$(2^{n^c})/O(n)$, which is a contradiction to the time hierarchy theorem for deterministic time. □

Now, Theorem 1.3 follows from above two lemmas.

Theorem 1.3 . *If* NP $\subseteq$ *quasi*$_{\mathrm{P}}$$-$QP *implies* NP $\subseteq$ QP*, then* NEXP $\not\subseteq$ P/poly.

Proof. By assumption, either NP $\subseteq \mathsf{QP}$ or NP $\nsubseteq quasi_{\mathrm{P}}{-}\mathsf{QP}$. In the first case, by Lemma 4.3, MA $\neq$ NEXP. In the second case also, by Lemma 4.5, MA $\neq$ NEXP. Thus, in either case, MA $\neq$ NEXP, which implies NEXP $\nsubseteq$ P/poly by Theorem 4.2.

5 Average-Case Circuit Lower Bounds Within PH

Kannan [Kan82] showed that for every k, there exist functions in the polynomial-time hierarchy for which no n^k-size circuits exist. However, this is a worst-case hardness result. Are there functions in PH that are hard on average for n^k-size circuits?

We show how to find such functions in the third level of the PH.

Theorem 5.1. *For any k and h, there is a language $L \in \mathrm{P}^{\Sigma_2^P}$ that is (n^k, n^h) hard.*

We first start with a function $g : \{0,1\}^{2(k+h)\log n} \to \{0,1\}$ that is (n^k, n^h) hard and then randomly pad the input to get a function f on n bits with the same hardness.

Theorem 5.2. *There is a function $g : \{0,1\}^{2(k+h)\log n} \to \{0,1\}$ that is (n^k, n^h) hard. Moreover, there is an $\mathrm{FP}^{\Sigma_2^P}$ procedure that outputs the lexicographically first such function.*

Proof. Consider a random function from $\{0,1\}^{2(k+h)\log n} \to \{0,1\}$ viewed as a Boolean string of length $n^{2(k+h)}$. Fix a circuit C of size n^k. The expected agreement between C and g is $\frac{n^{2(k+h)}}{2}$. Thus using Chernoff's bounds, $\Pr((C(x) = g(x)) > (1+\delta)\frac{n^{2(k+h)}}{2}) \leqslant e^{\frac{-\delta^2 n^{2(k+h)}}{6}}$. For $\delta = \frac{1}{n^h}$, this probability is $< 2^{-n^{2k}}$. There are at most $2^{n^{k+1}}$ circuits of size $\leqslant n^k$. Thus by union bound there exists a function $g : \{0,1\}^{2(k+h)\log n} \to \{0,1\}$ that is (n^k, n^h) hard.

Since the function is on $O(\log n)$ size inputs, it is easy to see that an $\mathrm{FP}^{\Sigma_2^P}$ procedure can output the lexicographically first such function.

Proof. (*Of Theorem 5.1*). Consider the function $f : \{0,1\}^n \to \{0,1\}$ defined as follows. Let $x = yz$ where y is the first $2(k+h)\log n$ bits of x. Define $f(x) = g(y)$ where g is the hard function from the above theorem. Claim is that the function f is (n^k, n^s) hard. For a contradiction, let D be a circuit of size at most n^k so that $\Pr_x((D(x) = f(x)) > \frac{1}{2} + \frac{1}{n^h})$. That is, $\Pr_{yz}((D(yz) = f(yz)) > \frac{1}{2} + \frac{1}{n^h})$. Then by an averaging argument there is a z so that $\Pr_y((D(yz) = f(yz)) > \frac{1}{2} + \frac{1}{n^h})$. Thus by hardwiring this z into D , we get a circuit D_z of size $\leqslant n^k$ so that $\Pr_y((D_z(y) = g(y)) > \frac{1}{2} + \frac{1}{n^h})$. This contradicts the hardness of g.

Theorem 1.4 is a special case of Theorem 5.1.

Acknowledgements

We thank the friendly staff of Iowa Western Community College, Atlantic, IA for providing facilities where a part of this work was done.

References

[Ale03] M. Alekhnovich. More on average case vs approximation complexity. In *Proceedings of 44th IEEE Symposium on Foundations of Computer Science*, pages 298–307, 2003.

[ART06] D. Achlioptas and F. Ricci-Tersenghi. On the solution-space geometry of random constraint satisfaction problems. In *Proceedings of Symposium on Theory of Computing*, page to appear, 2006.

[Ats06] A. Atserias. Non-uniform hardness for NP via black-box adversaries. In *Proceedings of Conference on Computational Complexity*, page to appear, 2006.

[BCK$^+$96] N. Bshouty, R. Cleve, S. Kannan, R. Gavalda, and C. Tamon. Oracles and queries that are sufficient for exact learning. *Journal of Computer and System Sciences*, 52:421–433, 1996.

[BFP05] H. Buhrman, L. Fortnow, and A. Pavan. Some results on derandomization. *Theory of Computing Systems*, 38(2):211–227, 2005.

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput.*, 13:850–864, 1984.

[BT03] A. Bogdanov and L. Trevisan. On worst-case to average-case reductions for np problems. In *Proceedings of the 44th IEEE Conference on Foundations of Computer Science*, pages 308–317, 2003.

[Cai01] J. Cai. $S_2^p \subseteq \mathrm{ZPP}^{\mathrm{NP}}$. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science, 2001*, pages 620–629, 2001.

[CW03] J. Cai and O. Watanabe. On proving circuit lower bounds against the polynomial hierarchy: Positive and negative results. In *Proceedings of Ninth Annual International Conference on Combinatorics and Computing*, pages 202–211, 2003.

[Fei02] Uriel Feige. Relations between average case complexity and approximation complexity. In *Proceedings of 35th Annual ACM Symposium on Theory of Computing*, pages 534–543, 2002.

[FPS03] L. Fortnow, A. Pavan, and S. Sengupta. Proving SAT does not have small circuits with an application to the two queries problem. In *Proceedings of the 18th IEEE Conference on Computational Complexity*, pages 347–350, 2003.

[Gol01] O. Goldreich. *Foundations of Cryptography - Volume 1.* Cambridge University Press, 2001.

[Gol04] O. Goldreich. *Foundations of Cryptography - Volume 2.* Cambridge University Press, 2004.

[GSTS05] D. Gutfreund, R. Shaltiel, and A. Ta-Shma. If NP languages are hard on the worst-case then it is easy to find their hard instances. In *IEEE Conference on Computational Complexity*, pages 243–257, 2005.

[Gur91] Y. Gurevich. Average case completeness. *Journal of Computer and System Sciences*, 42:346–398, 1991.

[IKW02] R. Impagliazzo, V. Kabanets, and A. Wigderson. In search of an easy witness: Exponential time vs Probabilistic polynomial time. *Journal of Computer and System Sciences*, 65:672–694, 2002.

[Imp95] R. Impagliazzo. A personal view of average-case complexity theory. In *Proceedings of the 10th Annual Conference on Structure in Complexity Theory*, pages 134–147. IEEE Computer Society Press, 1995.

[IW98] R. Impagliazzo and A. Wigderson. Randomness vs. time: de-randomization under a uniform assumption. In *39th Annual Symposium on Foundations of Computer Science: proceedings, 1998*, pages 734–743, 1998.

[JS05] J. Jackson and R. Servedio. On learning random DNF formulas under the uniform distribution. In *Proceedings of 9th International Workshop on Randomness and Computation*, pages 342–353, 2005.

[Kab01] V. Kabanets. Easiness assumptions and hardness tests: trading time for zero error. *Journal of Computer and System Sciences*, 63(2):236–252, 2001.

[Kan82] R. Kannan. Circuit-size lower bounds and non-reducibility to sparse sets. *Information and Control*, 55:40–56, 1982.

[Lau83] C. Lautemann. BPP and the polynomial hierarchy. *Information Processing Letters*, 17:215–217, November 1983.

[Lev86] L. Levin. Average case complete problems. *SIAM Journal of Computing*, 15:285–286, 1986.

[MVW99] P. B. Miltersen, N. V. Vinodchandran, and O. Watanabe. Super-polyomial versus half-exponential circuit size in the exponential hierarchy. In *Proceedings of Fifth Annual International Conference on Computing and Combinatorics*, pages 210–220, 1999.

[STV01] M. Sudan, L. Trevisan, and S. Vadhan. Pseudorandom generators without the XOR lemma. *JCSS: Journal of Computer and System Sciences*, 62, 2001.

[SU05] R. Shaltiel and C. Umans. Simple extractors for all min-entropies and a new pseudorandom generator. *Journal of the ACM*, 52, 2005.

[TV02] L. Trevisan and S. Vadhan. Pseudorandomness and average-case complexity via uniform reductions. In *Annual IEEE Conference on Computational Complexity*, volume 17, 2002.

[Uma02] C. Umans. Pseudo-random generators for all hardnesses. In *Symposium on Theory of Computing*, pages 627–634, 2002.

[Vio05] E. Viola. On constructing parallel pseudorandom generators from one-way functions. In *Proceedings of the 20th IEEE Conference on Computational Complexity*, 2005.

[vMS05] D. van Melkebeek and R. Santhanam. Holographic proofs and derandomization. *SIAM Journal on Computing*, 35(1):59–90, 2005.

[Yao82] A. Yao. Theory and applications of trapdoor functions. In *Proc. 23rd IEEE Symposium on Foundations of Computer Science*, pages 80–91, 1982.

Unbiased Rounding of Rational Matrices

Benjamin Doerr and Christian Klein

Max-Planck-Institut für Informatik, Saarbrücken, Germany

Abstract. Rounding a real-valued matrix to an integer one such that the rounding errors in all rows and columns are less than one is a classical problem. It has been applied to hypergraph coloring, in scheduling and in statistics. Here, it often is also desirable to round each entry randomly such that the probability of rounding it up equals its fractional part. This is known as unbiased rounding in statistics and as randomized rounding in computer science.

We show how to compute such an unbiased rounding of an $m \times n$ matrix in expected time $O(mnq^2)$, where q is the common denominator of the matrix entries. We also show that if the denominator can be written as $q = \prod_{i=1}^{\ell} q_i$ for some integers q_i, the expected runtime can be reduced to $O(mn \sum_{i=1}^{\ell} q_i^2)$. Our algorithm can be derandomised efficiently using the method of conditional probabilities.

Our roundings have the additional property that the errors in all initial intervals of rows and columns are less than one.

1 Introduction

In this paper, we analyze a rounding problem with strong connections to statistics, but also to different areas in discrete mathematics, computer science, and operations research. We present an efficient way to round a matrix to an integer one such that the rounding errors in all intervals (i.e., a set of consecutive entries) of rows and columns are small.

For real numbers a, b let $[a..b] := \{z \in \mathbb{Z} \mid a \leq z \leq b\}$. For $x \in \mathbb{R}$ let $\lfloor x \rfloor := \max\{z \in \mathbb{Z} \mid z \leq r\}$, $\lceil x \rceil := \min\{z \in \mathbb{Z} \mid z \geq r\}$ and $\{x\} := x - \lfloor x \rfloor$. For $q \in \mathbb{N}$ let $\frac{1}{q}\mathbb{Z} := \{\frac{p}{q} \mid p \in \mathbb{Z}\}$. We show the following.

Theorem 1. *For all $X \in \frac{1}{q}\mathbb{Z}^{m \times n}$ a randomized rounding $Y \in \mathbb{Z}^{m \times n}$ such that*

$$\forall b \in [1..n],\ i \in [1..m] : \left| \sum_{j=1}^{b} (x_{ij} - y_{ij}) \right| < 1, \tag{1}$$

$$\forall b \in [1..m],\ j \in [1..n] : \left| \sum_{i=1}^{b} (x_{ij} - y_{ij}) \right| < 1, \tag{2}$$

$$\left| \sum_{i=1}^{m} \sum_{j=1}^{n} (x_{ij} - y_{ij}) \right| < 1 \tag{3}$$

can be computed in expected time $O(mn \sum_{i=1}^{\ell} p_i^2)$, where $q = \prod_{i=1}^{\ell} p_i,\ p_i \in \mathbb{N}$ is a factorization of q.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 200–211, 2006.

The result above can be derandomised using the method of conditional probabilities, leading to a deterministic algorithm having asymptotically the same runtime.

Theorem 2. *For all $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$ a rounding $Y \in \mathbb{Z}^{m\times n}$ such that the inequalities* (1)*,* (2) *and* (3) *in Theorem 1 hold, can be computed in time $O(mn\sum_{i=1}^{\ell} p_i^2)$, where $q = \prod_{i=1}^{\ell} p_i$, $p_i \in \mathbb{N}$ is a factorization of q.*

Previous results on this particular rounding problem were given by Doerr et al. in [7]. Theorem 2 extends their result to arbitrary rational matrices. Their equivalent of Theorem 1, however, only works for matrices of numbers with finite binary expansion. Hence, they cannot, for example, round decimal fractions, as is often required in applications. For deterministic rounding they give an $O(mn\log(mn))$ time algorithm, while our algorithm is linear in the matrix size.

1.1 Baranyai's Rounding Lemma and Applications in Statistics

Baranyai [2] used a weaker variant of Theorem 2 to obtain his famous results on coloring and partitioning complete uniform hypergraphs. He showed that any matrix can be rounded such that the errors in all rows, all columns and the whole matrix are less than one. He used a formulation as flow problem to prove this statement, giving super-linear runtime. However, algorithmic issues were not his focus.

In statistics, Baranyai's result was independently obtained by Bacharach [1] (in a slightly weaker form), by Causey, Cox and Ernst [3], and, again independently, by Šíma [10]. There are two statistics applications for such rounding results. Note first that instead of rounding fractions to integers, our result also applies to rounding to multiples of any other integer (e.g., multiples of 10). Such a rounding can be used to improve the readability of data tables.

The main reason, however, to apply such a rounding procedure is confidentiality protection. Frequency counts that directly or indirectly disclose small counts may permit the identification of individual respondents. There are various methods to prevent this [12], one of which is *controlled rounding* [5]. Here, one tries to round an $(m+1)\times(n+1)$-table $\widetilde{X}$ given by

$$\begin{array}{c|c} (x_{ij})_{\substack{i=1\ldots m\\ j=1\ldots n}} & \left(\sum_{j=1}^{n} x_{ij}\right)_{i=1\ldots m} \\ \hline \left(\sum_{i=1}^{m} x_{ij}\right)_{j=1\ldots n} & \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} \end{array}$$

to an $(m+1)\times(n+1)$-table $\widetilde{Y}$ such that additivity is preserved, i.e., the last row and column of $\widetilde{Y}$ contain the associated totals of $\widetilde{Y}$. In our setting we round the $m\times n$-matrix X defined by the mn inner cells of the table $\widetilde{X}$ to obtain a controlled rounding.

The additivity in the rounded table allows to derive information on the row and column totals of the original table. In contrast to previous rounding algorithms, our result also permits to retrieve further reliable information from the rounded matrix, namely on the sums of consecutive elements in rows or columns. Such queries make sense if there is a linear ordering on statistical attributes.

Here is an example. Let x_{ij} be the number of people in country i that are j years old. Say Y is such that $\frac{1}{1000}Y$ is a rounding of $\frac{1}{1000}X$ as in Theorem 1. Now $\sum_{j=20}^{40} y_{ij}$ is the number of people in country i that are between 20 and 40 years old, apart from an error of less than 2000. Note that such guarantees are not provided by Baranyai [2], Bacharach [1], and Causey, Cox and Ernst [3].

1.2 Unbiased and Randomized Rounding

Our randomized algorithm has the additional property that each matrix entry is rounded up with probability equal to its fractional value. This is known as *randomized rounding* in computer science [9] and as *unbiased rounding* in statistics [4,8]. Here, a controlled rounding is computed such that the expected values of each table entry (including the totals) equals its fractional value in the original table.

Definition 1. *Let $x \in \mathbb{R}$. A random variable y is called* randomized rounding *of x, denoted by $y \approx x$, if* $\Pr(y = \lfloor x \rfloor + 1) = \{x\}$ *and* $\Pr(y = \lfloor x \rfloor) = 1 - \{x\}$. *For a matrix $X \in \mathbb{R}^{m\times n}$, we call a $\mathbb{Z}^{m\times n}$-valued random variable Y* randomized rounding *of X if $y_{ij} \approx x_{ij}$ for all $i \in [1..m], j \in [1..n]$.*

Note that if $y \approx x$, then $\Pr(|y-x| < 1) = 1$ and $\mathbb{E}(y) = x$. In fact, the converse holds as well. Hence we can restate Theorem 1 in the following stronger form.

Theorem 3. *For all $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$ a randomized rounding $Y \in \mathbb{Z}^{m\times n}$ fulfilling the additional constraints*

$$\begin{aligned} \forall b \in [1..n],\ i \in [1..m] : \sum_{j=1}^{b} x_{ij} &\approx \sum_{j=1}^{b} y_{ij}, \\ \forall b \in [1..m],\ j \in [1..n] : \sum_{i=1}^{b} x_{ij} &\approx \sum_{i=1}^{b} y_{ij}, \\ \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij} &\approx \sum_{i=1}^{m}\sum_{j=1}^{n} y_{ij} \end{aligned}$$

can be computed in expected time $O(mn\sum_{i=1}^{\ell} p_i^2)$, where $q = \prod_{i=1}^{\ell} p_i,\ p_i \in \mathbb{N}$ is a factorization of q.

2 Preliminaries

2.1 Random Walks

We need some well known facts about one-dimensional random walks with absorbing barriers. Consider a set of $n+1$ vertices labeled v_0 to v_n. From vertex v_i, $i \in [1..n-1]$, one can either take a step to vertex v_{i+1} or v_{i-1}, both with probability $\frac{1}{2}$. At the endpoints v_0 and v_n, no further steps can be taken. We write $\Pr(v_i \nearrow v_n)$ for the probability that a random walk from vertex v_i will reach v_n instead of v_0 and $\mathbb{E}(\mathrm{Steps}(v_i))$ for the expected number of steps a random walk starting in vertex v_i needs to reach either vertex v_0 or v_n

Lemma 1. $\Pr(v_i \nearrow v_n) = \frac{i}{n}$.

Proof. From the definition of random walks we obtain the equations $\Pr(v_n \nearrow v_n) = 1$, $\Pr(v_i \nearrow v_n) = \frac{1}{2}\Pr(v_{i-1} \nearrow v_n) + \frac{1}{2}\Pr(v_{i+1} \nearrow v_n)$, $i \in [1..n-1]$, and $\Pr(v_0 \nearrow v_n) = 0$. It can easily be checked that this system of equations has the unique solution $\Pr(v_i \nearrow v_n) = \frac{i}{n}$. □

Lemma 2. $\mathbb{E}(\mathrm{Steps}(v_i)) = i(n-i)$.

Proof. Again, we obtain the system of equations $\mathbb{E}(\mathrm{Steps}(v_0)) = \mathbb{E}(\mathrm{Steps}(v_n)) = 0$ and $\mathbb{E}(\mathrm{Steps}(v_i)) = 1 + \frac{1}{2}\mathbb{E}(\mathrm{Steps}(v_{i-1})) + \frac{1}{2}\mathbb{E}(\mathrm{Steps}(v_{i+1}))$, $i \in [1..n-1]$. It can easily be checked that this system of equations has the unique solution $\mathbb{E}(\mathrm{Steps}(v_i)) = i(n-i)$. □

2.2 Integrality of Row and Column Sums

In the following we always assume the input matrix X to be from $[0,1)^{m\times n}$. Otherwise, simply subtract the integral part of X before rounding and add it again afterwards. Furthermore, we assume X to have integral row and column sums, as justified by the following lemma from [7].

Lemma 3. *Assume that for any $X \in \mathbb{R}^{m\times n}$ with integral row and column sums, a rounding $Y \in \mathbb{Z}^{m\times n}$ satisfying inequality* (1) *and* (2) *from Theorem 1 can be computed in time $T(m,n)$. Then for all $\widetilde{X} \in \mathbb{R}^{m\times n}$ with arbitrary row and column sums, a rounding $\widetilde{Y} \in \mathbb{Z}^{m\times n}$ satisfying inequalities* (1), (2) *and* (3) *can be computed in time $T(m+1,n+1)+O(mn)$.*

3 Unbiased Rounding

3.1 Index Intervals

What properties does a rounding Y of X fulfilling the inequalities of Theorem 1 have? Substituting $b = n$ in inequality (1), we can deduce that the ith row of Y must contain exactly $\sum_{j=1}^{n} x_{ij}$ many 1-entries. To fulfill the inequality for $b \neq n$, there must be $\lfloor \sum_{j=1}^{b} x_{ij} \rfloor$ or $\lceil \sum_{j=1}^{b} x_{ij} \rceil$ many 1-entries in column 1 to b of the ith row of Y. Inequality (2) gives analogous statements for columns. This observation suggests that we should put one 1 in each interval bounded by two positions where the integral part of the partial row (resp. column) sum increases. This motivates the following definition.

We define the *kth index interval of the ith row of X* as

$$I_i^X(k) := \left\{ j \in [1..n] \mid x_{ij} \neq 0 \wedge \sum_{\ell=1}^{j} x_{i\ell} > k-1 \wedge \sum_{\ell=1}^{j-1} x_{i\ell} < k \right\}.$$

Column index intervals are defined analogous. Observe that the sum over all entries of an index interval is at least one. Because of this, each index interval consists of at least two non-zero elements. If the sum is more than one, the interval shares an entry with an neighboring interval. The following example shows a row of values and the corresponding index intervals.

$$\overbrace{0.2\quad 0.7\quad 0.}^{I(1)}\overbrace{8\quad 0.}^{I(2)}\overbrace{6\quad 0.4\quad 0.3}^{I(3)}\qquad\overbrace{0.5\quad 0.4\quad 0.1}^{I(4)}$$

The idea now is to "concentrate the total value of all entries" of an index interval into a single entry until it has value 1. For this, observe that if we pick two non-zero entries in the same row index interval and modify one by $+\frac{1}{q}$ and the other by $-\frac{1}{q}$, we don't change any of the partial sums left of the first or right of the second entry. In particular, the total sum of this row stays unchanged. The same holds for columns.

3.2 The Algorithm

The algorithm now iteratively modifies the matrix until all elements are 0 or 1. In each step it first constructs a cycle in the current matrix that alternatingly pairs two directly adjacent fractional elements in the same row interval resp. column interval[1]. This way each element of the cycle has one horizontal and one vertical neighbor in the cycle. How to construct such cycles will be discussed in Section 3.4. The algorithm then traverses this cycle and alternatingly adds $\frac{1}{q}$ and subtracts $\frac{1}{q}$ to each cycle entry.

```
ROUND($X \in (\frac{1}{q}\mathbb{Z} \cap [0,1))^{m\times n}$)
$t \leftarrow 0$
$X^{(0)} \leftarrow X$
Compute row and column index intervals of $X^{(0)}$
while $X^{(t)} \notin \{0,1\}^{m\times n}$
    do
        $C \leftarrow$ FINDCYCLE($X^{(t)}$)
        Choose $a \in \{+\frac{1}{q}, -\frac{1}{q}\}$
        $X^{(t+1)} \leftarrow$ alternatingly augment $X^{(t)}$ along $C$ by $\pm a$
        $t \leftarrow t+1$
        Update row and column index intervals of $X^{(t)}$.
return $Y := X^{(t)}$
```

Fig. 1. The rounding algorithm

The current matrix is stored in a two-dimensional doubly linked list where every non-integral entry has a pointer to the next non-integral entry in each direction. For the cycle finding step the algorithm must keep track of the index intervals of the current matrix $X^{(t)}$. To do this, the fractional parts $s_{ij}^{col} := \{\sum_{k=1}^{i} x_{kj}\}$ and $s_{ij}^{row} := \{\sum_{k=1}^{j} x_{ik}\}$ of the partial row and column sums of each entry $x_{ij}, i \in [1..m], j \in [1..n]$ are computed

[1] There is a special case where this is not true, as we will see later.

for the initial matrix and updated during the augmentation step. With these values the algorithm can decide if the neighbor of an entry belongs to the same index interval or not, based on the value of the neighbor entry and on the fractional part of the current entry. Whenever two neighboring elements inside the same index interval are augmented, only their partial sums change, hence the cost of an update is linear in the size of the current cycle.

If an augmentation changes an element to 0 or 1, it is removed from the data structure. Also, when updating the intervals, such element are ignored. By disregarding entries changed to 1, the corresponding row and column sums decrease by 1 and thus also the number of intervals decreases by 1 if this happens. Since the fractional part of an element that changes to 0 or 1 is also 0, this does not change the values s_{ij}^{col} or s_{ij}^{row} for any other element.

3.3 Runtime and Unbiasedness

For the moment let us assume that the call in Line 6 of the algorithm always returns a cycle and takes time proportional to the cycle size. Does the algorithm terminate? As we will see, this depends on how we choose a in Line 7. Each value for a corresponds to one of the two possible choices we have when doing the augmentation along the cycle. Either we start by adding $+\frac{1}{q}$ to the first element on the cycle, then $-\frac{1}{q}$ to the second and so on, or we start by adding $-\frac{1}{q}$ then $+\frac{1}{q}$ and so on. If one of this possibilities is chosen uniformly at random, we have the following theorem.

Theorem 4. *Assume that in Line 7 of the algorithm from Figure 1, a is chosen independently at random such that* $\Pr(a = \frac{1}{q}) = \Pr(a = -\frac{1}{q}) = \frac{1}{2}$. *Then the following holds.*

- *The algorithm terminates in expected time $O(mnq^2)$.*
- *Each $x_{ij}, i \in [1..m], j \in [1..n]$ is rounded to one with probability x_{ij}.*

Proof. Consider an element $x_{ij}, i \in [1..m], j \in [1..n]$ of the cycle. With probability $\frac{1}{2}$ each, we will either add or subtract $\frac{1}{q}$ from it. But this is equivalent to doing a random walk on a line with $q+1$ elements, starting from position $q \cdot x_{ij}$. From Lemma 2 it follows that the element becomes 0 or 1 after an expected number of $O(q^2)$ augmentations. As soon as this happens, x_{ij} will no longer belong to any index interval, and hence will no longer be chosen during the cycle construction. Since the matrix has mn entries, the first claim follows. The second claim follows immediately from Lemma 1. □

3.4 Finding Cycles

We now specify the function FINDCYCLE used by the algorithm to find a cycle along which it can round. As we will see, the fact that we aim at low errors in all initial intervals (and not only whole rows and columns) imposes some subtle additional difficulties.

First an arbitrary non-integral matrix entry a_1 is chosen as current entry. Then, alternatingly pick a non-integral entry directly adjacent to the current entry in the same row interval resp. in the same column interval as new current entry. This way, a sequence $(a_1, \ldots, a_\ell)$ of matrix entries is constructed. Since each index interval contains at least two fractional entries, the cycle construction routine can not fail to construct a cycle

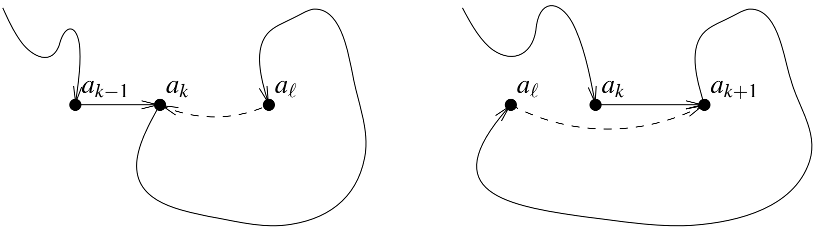


Fig. 2. The two possibilities during cycle construction

C as long as the matrix is not integral. The algorithm stops as soon as an element already picked before, say $a_k, k \in [1..\ell-1]$, can be chosen as current element. Assume that a_k and a_ℓ share a row interval[2]. By construction, either a_{k-1} or a_{k+1} will also be an element of this row. If a_{k-1} is an element of this row, $C := (a_k, \dots, a_\ell)$ is a cycle alternatingly pairing row and column elements sharing common intervals as needed by the main algorithm. However, if a_{k+1} is an element of this row, the above cycle would contain two successive edges pairing row entries, namely (a_ℓ, a_k) and (a_k, a_{k+1}).

In this case, the cycle $C := (a_{k+1}, \dots, a_\ell)$ is chosen instead which again alternatingly pairs row and column elements (See Figure 2 for details.). As this cycle now contains an edge pairing an element to its neighbors neighbor, the algorithm has to modify one additional partial sum during the augmentation, namely the one of a_k by $\pm\frac{1}{q}$ depending on how the pair (a_ℓ, a_{k+1}) is augmented. Observe that if a_k belongs to two overlapping index intervals, then a_ℓ and a_{k+1} belong to different intervals. As we will see in the analysis, this will not influence the correctness of the algorithm.

We finally argue that FINDCYCLE has an amortized runtime of $\Theta(|C|)$, where $|C|$ is the length of the cycle computed. Because augmenting along C only changes the local structure between two paired elements, the remaining elements of the sequence that were not chosen for C still alternatingly connect entries of the same row resp. column interval. Hence, the next time FINDCYCLE is called, it can reuse the part of the sequence not used to construct the cycle. Thus, over the whole algorithm, each element is touched during cycle construction as often as it is part of a cycle.

3.5 Correctness

In the following we only consider rows, as the arguments for columns are analogous. Hence, let $(x_1^{(t)}, \dots, x_n^{(t)}) := (x_{i1}^{(t)}, \dots, x_{in}^{(t)})$ be the elements of the ith row of $X^{(t)}$, for an arbitrary $i \in [1..m]$. Let $I^{(t)}(1), \dots, I^{(t)}(k)$ be the $k := \sum_{j=1}^{n} x_j^{(t)}$ index intervals of this row. For $\ell \in [1..k]$, we write $L(I^{(t)})(\ell) := \min(I^{(t)}(\ell))$ and $R(I^{(t)})(\ell) := \max(I^{(t)}(\ell))$ for the position of the leftmost, resp. rightmost entry of the ℓth interval. If $L(I^{(t)})(\ell)$ (resp. $R(I^{(t)})(\ell)$) does not belong to two intervals, we call it *proper*. If both $L(I^{(t)})(\ell)$ and $R(I^{(t)})(\ell)$ are proper, we call $I^{(t)}(\ell)$ *proper*.

The interior $I^{(t)}(\ell)^\circ$ of an interval is defined as the set of all elements that only belong to this interval. Hence, $I^{(t)}(\ell)^\circ = I^{(t)}(\ell)$ if and only if the interval is proper.

[2] Again the same holds for columns.

By the definition of index intervals, $R(I^{(t)})(\ell)$ is proper if and only if the partial sum up to this entry is integral. $L(I^{(t)})(\ell)$ is proper if and only if $R(I^{(t)})(\ell-1)$ is proper. Hence, $I^{(t)}(\ell)$ is proper if the sum over all entries in $I^{(t)}(\ell)$ is 1.

In the special case where we constructed a cycle pairing two entries $x_a^{(t)}, x_b^{(t)}$ from neighboring row intervals, those intervals share a common element $x_j^{(t)} \neq 0, j \in [a..b]$. Augmenting along this cycle introduces no inconsistencies in the columns, as all other pairs of entries are taken from a common interval. Modifying $x_a^{(t)}, x_b^{(t)}$ to, say, $x_a^{(t)} + \frac{1}{q}, x_b^{(t)} - \frac{1}{q}$, can, for the analysis, be viewed as modifying $x_a^{(t)} + \frac{1}{q}, x_j^{(t)} - \frac{1}{q}$ and $x_j^{(t)} + \frac{1}{q}, x_b^{(t)} - \frac{1}{q}$ independently. Since $x_j^{(t)}$ is a non-zero multiple of $\frac{1}{q}$ shared by both intervals, this is always possible.

First we show that as long as no element of an interval is set to one, the interval will only contract.

Lemma 4. *Let $I^{(t)}(\ell)$ be the ℓth interval at time t. Assume that no entry of $I^{(t)}(\ell)$ changes to* 1. *Let $I^{(t+1)}(\ell')$ be an interval at time $t+1$ that intersects $I^{(t)}(\ell)$.*

a) If $R(I^{(t)})(\ell)$ is proper, then $R(I^{(t+1)})(\ell')$ is proper and $R(I^{(t)})(\ell) \geq R(I^{(t+1)})(\ell')$.
b) If $L(I^{(t)})(\ell)$ is proper, then $L(I^{(t+1)})(\ell')$ is proper and $L(I^{(t)})(\ell) \leq L(I^{(t+1)})(\ell')$.
c) If $L(I^{(t)})(\ell)$ and $L(I^{(t+1)})(\ell')$ are not proper, then $L(I^{(t)})(\ell) = L(I^{(t+1)})(\ell')$.
d) If $L(I^{(t)})(\ell)$ is not proper, but $L(I^{(t+1)})(\ell')$ is proper, then
$R(I^{(t+1)})(\ell'-1) \leq R(I^{(t)})(\ell) = L(I^{(t)})(\ell) \leq L(I^{(t+1)})(\ell')$.

Proof. First observe that if $L(I^{(t)})(\ell)$ (resp. $R(I^{(t)})(\ell)$) is proper, it can only be paired with an element to its right (left). Since augmenting a pair does not change the partial sum of its right entry, we get the first statement. For the second statement observe that the partial sum up to $L(I^{(t)})(\ell)$ has the same fractional value as this element. Hence before it can be made small enough to belong to the $(\ell-1)$th interval, it will be zero, since all changes are done in steps of $\frac{1}{q}$. Statement c) follows from the fact that a shared element always has value larger than $\frac{1}{q}$. Since the augmentation only changes each value by at most $\frac{1}{q}$, this also proves d). □

Lemma 5. *Let $I^{(t)}(\ell)$ be the ℓth interval at time t and let $x_a^{(t)}$ be an element of $I^{(t)}(\ell)$ that changes to* 1. *If $x_a^{(t)}$ is shared with $I^{(t)}(\ell+1)$, both intervals will merge. Otherwise $I^{(t)}(\ell)$ vanishes and both the rightmost border of the interval to the left as well as the leftmost border of the interval to the right become proper.*

Proof. Surely $x_a^{(t)} = \frac{q-1}{q}$.

First, assume that $x_a^{(t)}$ is shared with $I^{(t)}(\ell+1)$. Then the partial sum for $x_a^{(t)}$ must have fractional value smaller than $x_a^{(t)}$, hence the intervals will merge.

Now assume that $x_a^{(t)}$ is an inner element of $I^{(t)}(\ell)$. Then the partial sum for $x_a^{(t)}$ must either be $\frac{q-1}{q}$ and $L(I^{(t)})(\ell) = a$ is proper, or it must be integral and $R(I^{(t)})(\ell) = a$ is proper. Note that in both cases the other border of $I^{(t)}(\ell)$ is the only non-integral element of this interval. If this element is also proper, the lemma obviously holds, hence assume it is shared (and thus has value at least $\frac{2}{q}$). If $a = L(I^{(t)})(\ell)$, then the augmentation will cause the partial sum for $x_a^{(t)}$ to become integral, making $R(I^{(t)})(\ell)$ the proper left border of $I^{(t)}(\ell+1)$. Otherwise the augmentation will cause the partial sum for $L(I^{(t)})(\ell)$ to become integral, making it the proper right border of $I^{(t)}(\ell-1)$. □

Lemma 6. *Let Y be a rounding of X computed by the algorithm in Figure 1. Then for each row there exists a bijective mapping between elements rounded to* 1 *and index intervals of this row in X, mapping each element to an interval containing it. The same holds for columns.*

Proof. Let $K := (I_a, \ldots, I_b)$ be a maximum collection of neighboring intervals in an arbitrary row of X such that $I_j \cap I_{j+1} \neq \emptyset$ for $j \in [a..(b-1)]$. In other words, exactly $L(I_a)$ and $R(I_b)$ are proper. Clearly, it suffices to prove the lemma for such subcollections.

First assume that at time t no element is changed to 1. If none of the shared borders of intervals in K become proper, then nothing changes according to Lemma 4. Otherwise, K decomposes into smaller collections of intervals which can be treated separately.

Now assume that at time t an inner element $x_j^{(t)}$ of a current interval changes to 1. By Lemma 5, this interval was obtained by merging $d-c+1$ neighboring intervals $I_c, \ldots, I_d, a \leq c \leq d \leq b$ of the initial collection. This means that their $d-c$ shared entries were set to 1 during the algorithm. Hence we can assign 1 to the interval of the initial collection containing $x_j^{(t)}$, and the remaining $d-c$ 1s to the other intervals. Since by Lemma 5 the borders of the neighbors of this interval become proper in this case, we get two smaller subcollections which can be treated separately. □

Theorem 5. *If Y is a rounding of X computed by the algorithm in Figure 1, then*

$$\forall b \in [1..n],\ i \in [1..m] : \left|\sum_{j=1}^{b}(x_{ij} - y_{ij})\right| < 1,$$

$$\forall b \in [1..m],\ j \in [1..n] : \left|\sum_{i=1}^{b}(x_{ij} - y_{ij})\right| < 1.$$

Proof. Let $b \in [1..n]$ and $i \in [1..m]$. If $x_{ib} = 0$ then $y_{ib} = 0$. Hence it suffices to regard the case $x_{ib} \neq 0$. Let $\ell \in \mathbb{N}$ be maximal such that x_{ib} is contained in the ℓth interval of the ith row of X. By definition, this means that $\ell - 1 < \sum_{j=1}^{b} x_{ij} \leq \ell$. By Lemma 6, $\ell - 1 \leq \sum_{j=1}^{b} y_{ij} \leq \ell$ holds. If $\sum_{j=1}^{b} x_{ij} < \ell$, this shows the theorem. In the other case, x_{ib} must be the last element of the ℓth interval, hence $\sum_{j=1}^{b} y_{ij} = \ell$. For columns, the proof is analogous. □

4 Iterative Rounding

If q has a non-trivial factorization $q = q_1 \cdot q_2$ with $q_1, q_2 \in \mathbb{N}_{\geq 2}$, this can be exploited to improve the runtime from Theorem 4. Our approach resembles that given by Doerr [6] for powers of 2.

```
ComputeRounding($X \in \frac{1}{q_1 \cdot q_2}\mathbb{Z}^{m\times n}$)
1  Compute $X' \in \frac{1}{q_1}\mathbb{Z}^{m\times n}, X'' \in \frac{1}{q_2}\mathbb{Z}^{m\times n}$ such that $X = X' + \frac{1}{q_1}X''$
2  $Y'' \leftarrow$ Round($X''$)$\in \{0,1\}^{m\times n}$
3  $\widetilde{X} \leftarrow X' + \frac{1}{q_1}Y'' \in \frac{1}{q_1}\mathbb{Z}^{m\times n}$
4  $Y \leftarrow$ Round($\widetilde{X}$)$\in \{0,1\}^{m\times n}$
5  return $Y \in \{0,1\}^{m\times n}$
```

Fig. 3. The factor rounding algorithm

Lemma 7. *Let $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$ be a rational matrix with $q = q_1 q_2$ and $q_1, q_2 \in \mathbb{N}$. Then* ComputeRounding *in Figure 3 will compute an unbiased rounding Y of X satisfying*

$$\forall b \in [1..n],\ i \in [1..m] : \left| \sum_{j=1}^{b} (x_{ij} - y_{ij}) \right| < 1,$$

$$\forall b \in [1..m],\ j \in [1..n] : \left| \sum_{i=1}^{b} (x_{ij} - y_{ij}) \right| < 1.$$

Proof. First note that the algorithm decomposes each matrix entry $x_{ij}, i \in [1..m], j \in [1..n]$ into $x'_{ij} \in \frac{1}{q_1}\mathbb{Z}$ and $x''_{ij} \in \frac{1}{q_2}\mathbb{Z}$. To show unbiasedness, observe that in Line 2, an unbiased rounding $y''_{ij} \in \{0,1\}$ of x''_{ij} is computed according to Theorem 4. In other words, $\widetilde{x}_{ij}$ computed in Line 3 will have value $x'_{ij} + \frac{1}{q_1}$ with probability x'', and value x'_{ij} otherwise. From Line 4 it follows that

$$\begin{aligned}
\Pr(y_{ij} = 1) = \Pr(\widetilde{x}_{ij} \nearrow 1) &= x''_{ij} Pr((x'_{ij} + \tfrac{1}{q_1}) \nearrow 1) + (1 - x''_{ij}) Pr(x'_{ij} \nearrow 1) \\
&= x''_{ij}(x'_{ij} + \tfrac{1}{q_1}) + (1 - x''_{ij}) x'_{ij} \\
&= \tfrac{1}{q_1} x''_{ij} + x'_{ij} = x_{ij},
\end{aligned}$$

hence the algorithm computes an unbiased rounding of X.

To see that the rounding computed in Figure 3 is a controlled rounding satisfying our additional constraints, let $s_{ij}(X) := \sum_{k=1}^{i} x_{kj}$ for $i \in [1..m], j \in [1..n]$, be the sum over the first i elements of the jth column of X. In Line 2, a controlled rounding Y'' of X'' satisfying our additional constraints is computed, hence $|s_{ij}(X'' - Y'')| \leq 1 - \frac{1}{q_2}$. A similar statement holds for Y and $\widetilde{X}$ in Line 4, namely $|s_{ij}(\widetilde{X} - Y)| \leq 1 - \frac{1}{q_1}$. These two error bounds and the triangle inequality yield

$$\begin{aligned}
|s_{ij}(X - Y)| &= |s_{ij}(X' + \tfrac{1}{q_1}X'' - \tfrac{1}{q_1}Y'' + \tfrac{1}{q_1}Y'' - Y)| \\
&\leq |s_{ij}(\widetilde{X} - Y)| + \tfrac{1}{q_1}|s_{ij}(X'' - Y'')| \\
&\leq 1 - \tfrac{1}{q_1} + \tfrac{1}{q_1}(1 - \tfrac{1}{q_2}) = 1 - \tfrac{1}{q},
\end{aligned}$$

hence the error in all initial column intervals is at most $1-\frac{1}{q}$. The proof for the error in initial row intervals and in single elements is analogous. □

Now let $q=\prod_{i=1}^{\ell} q_i,\ q_i \in \mathbb{N}$ be a factorization of the denominator of X. Then the algorithm in Figure 3 can be applied recursively to get the main result as stated in Theorem 1.

Since for $X \in \{0,\frac{1}{2}\}^{m\times n}$, an augmentation of an element by $\pm\frac{1}{2}$ will always change the element to either 0 or 1, the algorithm from Figure 1 will run in *deterministic* time $O(mn)$ for this special case. Using this observation and choosing $q=2^{\ell}$, this gives the result from [7] for unbiased rounding of matrices of ℓ-bit numbers.

Corollary 1. *Let $X \in [0,1)^{m\times n}$ be a matrix of ℓ-bit numbers. Then an unbiased controlled rounding of X satisfying equations* (1)*,* (2) *and* (3) *from Theorem 1 can be computed in time $O(mn\ell)$.*

5 Derandomisation

The algorithm in Figure 1 can be derandomised using the method of conditional probabilities (cf. [11]). For this, observe that by Lemma 2 the expected number $\mathbb{E}(\mathrm{Steps}(X))$ of augmentations needed to round a given matrix $X \in (\frac{1}{q}\mathbb{Z}\cap[0,1))^{m\times n}$ is

$$\mathbb{E}(\mathrm{Steps}(X)) = \sum_{i=1}^{m}\sum_{j=1}^{n} x_{ij}(q-x_{ij}) = O(mnq^2).$$

The derandomisation now works as follows. At the beginning of the algorithm in Figure 1, $\mathbb{E}(\mathrm{Steps}(X))$ is computed. Each time one of the two possible ways to augment along a cycle C in Line 7 of the algorithm must be chosen, this isn't done randomly. Instead, the augmentation for which the algorithm would need the fewer number of expected steps if it would continue choosing randomly is picked. By Lemma 2 it follows that

$$\mathbb{E}(\mathrm{Steps}(X)) = |C| + \tfrac{1}{2}\mathbb{E}(\mathrm{Steps}(X-X_C)) + \tfrac{1}{2}\mathbb{E}(\mathrm{Steps}(X+X_C)),$$

where X_C is the matrix for one of the two possible augmentations along C. From this formula it follows that $\mathbb{E}(\mathrm{Steps}(X-X_C))$ and $\mathbb{E}(\mathrm{Steps}(X+X_C))$ cannot both be larger than $\mathbb{E}(\mathrm{Steps}(X))-|C|$. Hence, each time the algorithm augments along a cycle C, $\mathbb{E}(\mathrm{Steps}(X))$ decreases by at least $|C|$, since the augmentation giving the smaller expected value is picked.

Calculating $\mathbb{E}(\mathrm{Steps}(X))$ for the input matrix needs time $O(mn)$. Deciding which augmentation to use for cycle C in step t can be done in time $O(|C|)$ while constructing the cycle. The value $\mathbb{E}(\mathrm{Steps}(X^{(t)}))$ can be derived from $\mathbb{E}(\mathrm{Steps}(X^{(t-1)}))$ in $O(|C|)$ time while doing the actual augmentation. This gives the following theorem.

Theorem 6. *For all $X \in \frac{1}{q}\mathbb{Z}^{m\times n}$ a rounding $Y \in \mathbb{Z}^{m\times n}$ such that the inequalities* (1)*,* (2) *and* (3) *from Theorem 1 hold can be computed in time $O(mnq^2)$.*

Together with Lemma 7, this yields Theorem 2 from the introduction.

References

1. M. Bacharach. Matrix rounding problems. *Management Science (Ser. A)*, 12:732–742, 1966.
2. Zs. Baranyai. On the factorization of the complete uniform hypergraph. In *Infinite and finite sets (Colloq., Keszthely, 1973; dedicated to P. Erdős on his 60th birthday), Vol. I*, pages 91–108. Colloq. Math. Soc. Jánōs Bolyai, Vol. 10. North-Holland, Amsterdam, 1975.
3. B. D. Causey, L. H. Cox, and L. R. Ernst. Applications of transportation theory to statistical problems. *Journal of the American Statistical Association*, 80:903–909, 1985.
4. L. H. Cox. A constructive procedure for unbiased controlled rounding. *Journal of the American Statistical Association*, 82:520–524, 1987.
5. L. H. Cox and L. R. Ernst. Controlled rounding. *Informes*, 20:423–432, 1982.
6. B. Doerr. Generating randomized roundings with cardinality constraints and derandomizations. In *23rd Annual Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 571–583, Marseille, France, 2006. Springer.
7. B. Doerr, T. Friedrich, C. Klein, and R. Osbild. Unbiased matrix rounding. In *10th Scandinavian Workshop on Algorithm Theory*, volume 4059 of *Lecture Notes in Computer Science*, pages 102–112, Riga, Latvia, 2006. Springer.
8. I. P. Fellegi. Controlled random rounding. *Survey Methodology*, 1:123–133, 1975.
9. P. Raghavan. Probabilistic construction of deterministic algorithms: Approximating packing integer programs. *J. Comput. Syst. Sci.*, 37:130–143, 1988.
10. J. Šíma. Table rounding problem. *Computers and Artificial Intelligence*, 18:175–189, 1999.
11. J. Spencer. Randomization, derandomization and antirandomization: Three games. *Theoretical Computer Science*, 131:415–429, 1994.
12. L. Willenborg and T. de Waal. *Elements of Statistical Disclosure Control*, volume 155 of *Lecture Notes in Statistics*. Springer, 2001.

Rational Behaviour and Strategy Construction in Infinite Multiplayer Games[*]

Michael Ummels

Mathematische Grundlagen der Informatik, RWTH Aachen, Germany
ummels@informatik.rwth-aachen.de

Abstract. We study infinite games played by arbitrarily many players on a directed graph. Equilibrium states capture rational behaviour in these games. Instead of the well-known notion of a Nash equilibrium, we focus on the notion of a subgame perfect equilibrium. We argue that the latter one is more appropriate for the kind of games we study, and we show the existence of a subgame perfect equilibrium in any infinite game with ω-regular winning conditions.

As, in general, equilibria are not unique, it is appealing to compute one with a maximal payoff. This problem corresponds naturally to the problem of deciding given a game and two payoff vectors whether the game has an equilibrium with a payoff in between the given thresholds. We show that this problem is decidable for games with ω-regular winning conditions played on a finite graph and analyse its complexity. Moreover, we establish that any subgame perfect equilibrium of a game with ω-regular winning conditions played on a finite graph can be implemented by finite-state strategies.

Finally, we consider logical definability. We state that if we fix the number of players together with an ω-regular winning condition for each of them and two payoff vectors the property that a game has a subgame perfect equilibrium with a payoff in between the given thresholds is definable in the modal μ-calculus.

1 Introduction

We study *infinite games of perfect information* [5] played by multiple players on a directed graph. Intuitively, a *play* of such a game evolves by moving a token along edges of the graph. Every vertex of the graph is controlled by precisely one player. Whenever the token arrives at some vertex, the player who controls this vertex must move the token to a successor vertex. Thus a play of such a game is an infinite path through the graph. Plays are mapped to *payoffs*, one for each player. In the simplest case, which we discuss here, payoffs are just 0 and 1, i.e. each player either wins or loses a given play of the game. We allow, however, that a play is won by more than one player or even by no player at all.

[*] This research has been partially supported by the European Community Research Training Network "Games and Automata for Synthesis and Validation" (GAMES).

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 212–223, 2006.

Infinite games have been successfully applied in the verification and synthesis of reactive systems. Such a system is usually modelled as a game between the system and its environment where the environment's objective is the complement of the system's objective, so the environment is considered hostile. Therefore, traditionally, the research in this area has mostly looked at two-player games where each play is won by precisely one of the two players, so-called *two-player zero-sum games*. However, motivated by the modelling of distributed systems, interest in the general case has increased in recent years [1,2].

Example 1. Consider a scenario where three agents are competing for a resource that can only be used by at most two of them using the following protocol: At first, agent 1 decides whether to grant the other agents 2 and 3 access to the resource or to pass control to agent 2. If control is passed to agent 2, she can decide whether to share access to the resource with agent 1 or to grant agent 3 exclusive access to the resource. The situation is naturally modelled by the following game with its arena depicted in Fig. 1; round vertices are controlled by player 1; boxed vertices are controlled by player 2; player 3 does not control any vertex; player 1 wins if vertex 5 is visited (infinitely often); player 2 wins if vertex 4 or vertex 5 is visited (infinitely often); player 3 wins if vertex 3 or vertex 4 is visited (infinitely often); the initial vertex is 1.

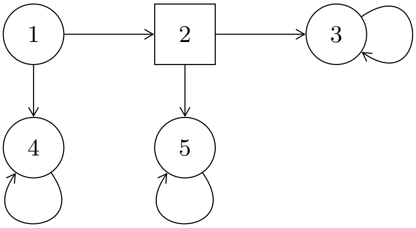


Fig. 1. A game with three players

Different *solution concepts* [13] have been proposed to model rational behaviour in games. The classical solution concept offered by game theory is the one of a *Nash equilibrium* [12]. In a Nash equilibrium no player can receive a better payoff by unilaterally changing her strategy. For instance, the game described in Example 1 has two Nash equilibrium payoffs:

1. Players 1 and 2 win; a Nash equilibrium with this payoff is the combination of strategies where player 1 moves from vertex 1 to vertex 2 and player 2 moves from vertex 2 to vertex 5.
2. Player 3 wins; a Nash equilibrium with this payoff is the combination of strategies where player 1 moves from vertex 1 to vertex 4 and player 2 moves from vertex 2 to vertex 3.

Intuitively, the second equilibrium is not rational because if player 1 moved from vertex 1 to vertex 2 instead player 2 should change her strategy and move to vertex 5 instead because this is then the only way for her to win. However, in the definition of a Nash equilibrium it is not taken into account that players can change their strategies during a play.

An equilibrium concept that respects this possibility is the notion of a subgame perfect equilibrium [16]. For being a subgame perfect equilibrium, a choice of strategies is not only required to be optimal for the initial vertex but for every possible initial history of the game (including histories not reachable in the equilibrium play). In the example the second Nash equilibrium is not a subgame perfect equilibrium because moving from vertex 2 to vertex 3 is not optimal for player 2 after the play has reached vertex 2.

Subgame perfect equilibria have been well studied in the context of *finite games*. In particular, Kuhn [7] showed that every finite game has a subgame perfect equilibrium. Yet, we think that the concept is also worth to be analysed in the context of infinite games because the possibility of changing strategies during a play is not unique to finite games. In this paper we show the existence of subgame perfect equilibria for infinite games with parity winning conditions, a standard form of ω-regular winning conditions, and we remark that the same holds for the greater class of Borel objectives. This generalises a result by Chatterjee et al. [2] about the existence of Nash equilibria in infinite games. Based on the proof, we also develop an algorithm for computing a subgame perfect equilibrium of a game with parity winning conditions.

We then turn to the potentially harder problem of finding a subgame perfect equilibrium with a maximal payoff. This problem is closely related to the problem of deciding given a game and two payoff vectors whether the game has a subgame perfect equilibrium with a payoff in between the given thresholds. Using a translation into tree automata, we show that the latter problem is decidable for games with ω-regular winning conditions played on a finite graph. In particular, we show that for games with Rabin objectives the problem is decidable in exponential time in general and in polynomial time if the number of players and the number of Rabin pairs are bounded. Moreover, we show that the problem is 2EXPTIME-complete for games with LTL objectives.

Naturally, we are also interested in the complexity of strategies realising an equilibrium. We show that for games with ω-regular winning conditions played on a finite graph any subgame perfect equilibrium can be implemented by finite-state strategies. This is the best one can hope for because, even for games with Büchi objectives, positional strategies, in general, do not suffice to implement any Nash or subgame perfect equilibrium.

We conclude this paper with a section on logical definability. It is well known that for any fixed number m the property that the first player wins a two-player zero-sum parity game with m different priorities is definable in the modal μ-calculus. We state a natural generalisation of this fact for multiplayer games: If we fix the number of players together with an ω-regular winning condition for each of them and two payoff vectors, the property that a game has a subgame perfect equilibrium with a payoff in between the given thresholds is definable in the modal μ-calculus as well.

2 Infinite Multiplayer Games

The definition of an infinite (two-player zero-sum) game played on a directed, coloured graph [19] easily generalises to the multiplayer setting. Formally, we define an *infinite multiplayer game* as a tuple $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, \chi, (W_i)_{i\in\Pi})$ where

- Π is a finite set of *players*;
- (V, E) is a (not necessarily finite) directed graph;
- $(V_i)_{i\in\Pi}$ is a partition of V;
- $\chi : V \to C$ for some set C;
- $W_i \subseteq C^\omega$ for all $i \in \Pi$.

The structure $G = (V, (V_i)_{i\in\Pi}, E, \chi)$ is called the *arena* of $\mathcal{G}$; χ is called the *colouring* of G, and W_i is called the *winning condition* of player $i \in \Pi$. For the sake of simplicity, we assume that $uE := \{v \in V : (u, v) \in E\} \neq \emptyset$ for all $u \in V$, i.e. each vertex of G has at least one outgoing edge. We say that $\mathcal{G}$ is *finitely coloured* if $\chi : V \to C$ for a finite set C. Finally, we call $\mathcal{G}$ a *zero-sum game* if the sets W_i define a partition of V^ω. Thus if $\mathcal{G}$ is an infinite two-player zero-sum game with players 0 and 1 it suffices to define V_0 and W_0, and we just write $\mathcal{G} = (V, V_0, E, \chi, W_0)$.

A *play* or *history* of $\mathcal{G}$ is an infinite or finite path in G, respectively. We say that a play π is *won* by player $i \in \Pi$ if $\chi(\pi) \in W_i$. The *payoff* of a play π of $\mathcal{G}$ is the vector $\mu(\pi) \in \{0, 1\}^\Pi$ defined by $\mu(\pi)_i = 1$ if π is won by player i. A *strategy of player i in $\mathcal{G}$* is a total function $\sigma : V^*V_i \to V$ assigning to each nonempty sequence wv of vertices ending in a vertex v of player i another vertex $\sigma(wv)$ such that $(v, \sigma(wv)) \in E$. We say that a play π of $\mathcal{G}$ is *consistent* with a strategy σ of player i if $\pi(k + 1) = \sigma(\pi(0)\dots\pi(k))$ for all $k < \omega$ with $\pi(k) \in V_i$. A *strategy profile of* $\mathcal{G}$ is a tuple $(\sigma_i)_{i\in\Pi}$ where σ_i is a strategy of player i in $\mathcal{G}$.

A strategy σ of player i in $\mathcal{G}$ is called *positional* if σ depends only on the current vertex, i.e. if $\sigma(wv) = \sigma(v)$ for all $w \in V^*$ and $v \in V_i$. More generally, σ is called a *finite-state strategy* if the equivalence relation $\sim_\sigma$ on V^* defined by $w \sim_\sigma w'$ if $\sigma(wz) = \sigma(w'z)$ for all $z \in V^*V_i$ has finite index. In other words, a finite-state strategy is a strategy that can be implemented by a finite automaton with output. A strategy profile $(\sigma_i)_{i\in\Pi}$ of $\mathcal{G}$ is called *positional* or a *finite-state strategy profile* if each σ_i is positional or a finite-state strategy, respectively.

It is sometimes convenient to designate an initial vertex $v_0 \in V$ of the game. We call the tuple $(\mathcal{G}, v_0)$ an *initialised infinite multiplayer game.* A *play (history) of* $(\mathcal{G}, v_0)$ is a play (history) of $\mathcal{G}$ starting with v_0. A strategy (strategy profile) of $(\mathcal{G}, v_0)$ is just a strategy (strategy profile) of $\mathcal{G}$. A strategy σ of some player i in $(\mathcal{G}, v_0)$ is *winning* if every play of $(\mathcal{G}, v_0)$ consistent with σ is won by player i. A strategy profile $(\sigma_i)_{i\in\Pi}$ of $(\mathcal{G}, v_0)$ determines a unique play of $(\mathcal{G}, v_0)$ consistent with each σ_i, called the *outcome of* $(\sigma_i)_{i\in\Pi}$ and denoted by $\langle(\sigma_i)_{i\in\Pi}\rangle$ or, in the case that the initial vertex is not understood from the context, $\langle(\sigma_i)_{i\in\Pi}\rangle_{v_0}$. In the following we will often use the term *game* to denote an *(initialised) infinite multiplayer game.*

For a game $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, \chi, (W_i)_{i\in\Pi})$ and a history h of $\mathcal{G}$, let the game $\mathcal{G}|_h = (\Pi, V, (V_i)_{i\in\Pi}, E, \chi, (W_i|_h)_{i\in\Pi})$ be defined by $W_i|_h = \{\alpha \in C^\omega : \chi(h)\cdot\alpha \in W_i\}$. For an initialised game $(\mathcal{G}, v_0)$ and a history hv of $(\mathcal{G}, v_0)$, we call the initialised game $(\mathcal{G}|_h, v)$ the *subgame of* $(\mathcal{G}, v_0)$ *with history* hv. For a strategy σ of player $i \in \Pi$ in $\mathcal{G}$, let $\sigma|_h : V^*V_i \to V$ be defined by $\sigma|_h(wv) = \sigma(hwv)$. Obviously, $\sigma|_h$ is a strategy of player i in $\mathcal{G}|_h$.

A strategy profile $(\sigma_i)_{i\in\Pi}$ of a game $(\mathcal{G}, v_0)$ is called a *Nash equilibrium* if for any player $i \in \Pi$ and all her possible strategies σ' in $(\mathcal{G}, v_0)$ the play $\langle\sigma', (\sigma_j)_{j\in\Pi\setminus\{i\}}\rangle$ is won by player i only if the play $\langle(\sigma_j)_{j\in\Pi}\rangle$ is also won by her. The strategy profile $(\sigma_i)_{i\in\Pi}$ is called a *subgame perfect equilibrium (SPE)* if $(\sigma_i|_h)_{i\in\Pi}$ is a Nash equilibrium of $(\mathcal{G}|_h, v)$ for every history hv of $(\mathcal{G}, v_0)$.

Winning conditions. We have introduced winning conditions as abstract sets of infinite sequences over the set of colours. In verification winning conditions are usually *ω-regular sets* specified by formulae of the logic S1S (monadic second-order logic on infinite words) or LTL (linear-time temporal logic) referring to unary predicates P_c indexed by the set C of colours, which is assumed to be finite. Special cases are the following well-studied winning conditions:

- *Büchi* (given by $F \subseteq C$): defines the set of all $\alpha \in C^\omega$ such that $\alpha(k) \in F$ for infinitely many $k < \omega$.
- *Parity* (given by a *priority function* $\Omega : C \to \omega$): defines the set of all $\alpha \in C^\omega$ such that the least number occurring infinitely often in $\Omega(\alpha)$ is even.
- *Rabin* (given by a set Ω of *pairs* (G_i, R_i) where $G_i, R_i \subseteq C$): defines the set of all $\alpha \in C^\omega$ such that there exists an index i such that $\alpha(k) \in G_i$ for infinitely many $k < \omega$ but $\alpha(k) \in R_i$ only for finitely many $k < \omega$.

Note that the Büchi condition is a special case of the parity condition with two priorities and that the parity condition is a special case of the Rabin condition. Also note that Büchi, parity and Rabin conditions are *prefix independent*, i.e. for every $\alpha \in C^\omega$ and $w \in C^*$ it is the case that α satisfies the condition if and only if $w\alpha$ does.

We call a finitely coloured game $\mathcal{G}$ a *multiplayer S1S, LTL, Büchi, parity or Rabin game* if the winning condition of each player is of type S1S, LTL, Büchi, parity or Rabin, respectively.[1] Any of these games is called an *ω-regular game.* It is well known that the complement of a Rabin condition is again expressible as a Rabin condition if and only if it is equivalent to a parity condition. Thus any two-player zero-sum Rabin game is also a two-player zero-sum parity game. Observe that $\mathcal{G}|_h = \mathcal{G}$ for every history h of $\mathcal{G}$ if $\mathcal{G}$ is a multiplayer Büchi, parity or Rabin game because the winning conditions in these games are prefix independent.

We say that two initialised games $(\mathcal{G}, v_0)$ and $(\mathcal{G}', v_0')$ are *equivalent* if for any (finite-state) Nash or subgame perfect equilibrium of $(\mathcal{G}, v_0)$ there exists a

[1] Our notation differs here from the usual notation for two-player zero-sum games where a Büchi or Rabin game is a game where the winning condition of the first player is a Büchi or Rabin condition, respectively.

(finite-state) Nash or subgame perfect equilibrium of $(\mathcal{G}', v_0')$, respectively, with the same payoff and, vice versa, for any (finite-state) Nash or subgame perfect equilibrium of $(\mathcal{G}', v_0')$ there exists a (finite-state) Nash or subgame perfect equilibrium of $(\mathcal{G}, v_0)$, respectively, with the same payoff. As for two-player zero-sum games (see, for example, [17]), any ω-regular multiplayer game can be reduced to an equivalent multiplayer parity game.

Proposition 1. *Any ω-regular multiplayer game $(\mathcal{G}, v_0)$ is equivalent to a multiplayer parity game $(\mathcal{G}', v_0')$. If $\mathcal{G}$ is a multiplayer LTL game with k players, n vertices and winning conditions of size $\leq m$, then $\mathcal{G}'$ has $n \cdot 2^{2^{O(m)+\log k}}$ vertices and $2^{O(m)}$ priorities for each player.*

3 Existence of Subgame Perfect Equilibria

The aim of this section is to show that any ω-regular multiplayer game has a subgame perfect equilibrium. By Proposition 1, it suffices to consider multiplayer parity games. In the case of two-player zero-sum games, parity games are *positionally determined*, i.e. one of the two players not only has a winning strategy but a positional one.

Theorem 2 (Emerson-Jutla [3], Mostowski [10]). *Two-player zero-sum parity games are positionally determined.*

Moreover, positional winning strategies can always be chosen uniformly, i.e. independently of the initial vertex (see, for example, [19]). Hence any two-player zero-sum parity game has a positional subgame perfect equilibrium.

Corollary 3. *Any two-player zero-sum parity game has a positional subgame perfect equilibrium.*

Using Martin's determinacy theorem for two-player zero-sum Borel games [8], Chatterjee et al. [2] showed that any multiplayer game with Borel winning conditions has a Nash equilibrium. Rephrased for parity games, roughly speaking, their proof goes as follows: Given a multiplayer parity game $(\mathcal{G}, v_0)$, for each player i, consider the two-player zero-sum parity game $(\mathcal{G}_i, v_0)$ where player i plays against the coalition of all other players. By Corollary 3, this game has a subgame perfect equilibrium consisting of a strategy σ_i of player i and a strategy σ_{-i} of the coalition, i.e. for every vertex v of $\mathcal{G}$ either σ_i or σ_{-i} is winning in $(\mathcal{G}_i, v)$. In the equilibrium player i plays her strategy σ_i as long as no other player j deviates from her strategy σ_j in which case she switches to the coalition strategy σ_{-j}. In game theory this type of strategy is known under the term "threat strategy" and has its origin in the theory of repeated games (cf. [13, Chapter 8]). To make the Nash equilibrium a subgame perfect equilibrium, we do not consider threat strategies in the original game but in a game arising as a fixed point of a deflationary operator defined on the original game.

Theorem 4. *Any multiplayer parity game has a subgame perfect equilibrium.*

Proof. Let $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, \chi, (\Omega_i)_{i\in\Pi})$ be a multiplayer parity game. For each ordinal α we define a set $E^\alpha \subseteq E$ beginning with $E^0 = E$ and

$$E^\lambda = \bigcap_{\alpha<\lambda} E^\alpha$$

for limit ordinals λ. To define $E^{\alpha+1}$ from E^α, we consider for each player $i \in \Pi$ the two-player zero-sum parity game $\mathcal{G}_i^\alpha = (V, V_i, E^\alpha, \chi, \Omega_i)$ where player i plays against the coalition of all other players. By Corollary 3, we can fix a positional subgame perfect equilibrium $(\sigma_i^\alpha, \sigma_{-i}^\alpha)$ of this game where σ_i^α is a strategy of player i and σ_{-i}^α is a strategy of the coalition. Let X_i^α be the set of all $v \in V$ such that σ_i^α is winning in $(\mathcal{G}_i^\alpha, v)$. For vertices $v \in V_i \cap X_i^\alpha$ we delete all outgoing edges except the one taken by the strategy σ_i^α, i.e. we define

$$E^{\alpha+1} = E^\alpha \setminus \bigcup_{i\in\Pi} \{(u,v) \in E : u \in V_i \cap X_i^\alpha \text{ and } v \neq \sigma_i^\alpha(u)\} \ .$$

Obviously, the sequence $(E^\alpha)_{\alpha\in\mathrm{On}}$ is nonincreasing. Thus we can fix the least ordinal ξ with $E^\xi = E^{\xi+1}$ and define $\sigma_i = \sigma_i^\xi$ and $\sigma_{-i} = \sigma_{-i}^\xi$. Moreover, for each player $j \neq i$ let $\sigma_{j,i}$ be the positional strategy of player j in $\mathcal{G}$ that is induced by σ_{-i}. Player i's equilibrium strategy τ_i is defined as follows: Player i plays σ_i as long as no other player deviates. Whenever some player $j \neq i$ deviates from her equilibrium strategy τ_j, player i switches to $\sigma_{i,j}$. Then $(\tau_i)_{i\in\Pi}$ is a subgame perfect equilibrium of $(\mathcal{G}, v_0)$ for any initial vertex v_0. □

More generally, Theorem 4 holds for games with (quasi-)Borel winning conditions [9]. The proof is similar to the proof for parity games but based on Martin's determinacy theorem for (quasi-)Borel sets [8,9]. However, Martin's theorem can only guarantee the existence of an arbitrary subgame perfect equilibrium in a two-player zero-sum game with (quasi-)Borel winning conditions, not necessarily a positional one. To ensure that the proof works, we have to assume that the arena of the game under consideration is a forest. Over a forest any strategy is obviously equivalent to a positional one. The justification for this assumption is that we can always replace the arena of an arbitrary game by its unravelling from the initial vertex, ending up in an equivalent game. See [18, Chapter 3] for the full proof.

Theorem 5. *Any multiplayer game with (quasi-)Borel winning conditions has a subgame perfect equilibrium.*

Naturally, we are interested in the complexity of strategies realising a subgame perfect equilibrium. It is easy to see that for parity games played on a finite arena the subgame perfect equilibrium constructed in the proof of Theorem 4 is, in fact, a finite-state one. This leaves open the existence of a positional subgame perfect equilibrium as it is guaranteed in the two-player zero-sum case (even for games with an infinite arena). We are only able to give a partial answer to this question, namely in the case of only two players 1 and 2. Indeed, it is easy to see that the positional strategies $\sigma_{1,2}$ and $\sigma_{2,1}$ as defined in the proof of Theorem 4 form a subgame perfect equilibrium in this case.

Theorem 6. *Any two-player parity game has a positional subgame perfect equilibrium.*

A simple algorithm. Knowing that there always exists a subgame perfect equilibrium in an ω-regular multiplayer game, the next challenge is to compute one. Algorithm 1 is a simple procedure for computing a subgame perfect equilibrium of a multiplayer parity game derived from the proof of Theorem 4 in a straightforward way. Thus its correctness follows immediately.

Algorithm 1. Computing a finite-state SPE of a multiplayer parity game.

```
input multiplayer parity game $\mathcal{G} = (\Pi, V, (V_i)_{i\in\Pi}, E, \chi, (\Omega_i)_{i\in\Pi})$
$E_{\text{new}} := E$
repeat
    $E_{\text{old}} := E_{\text{new}}$
    for each $i \in \Pi$ do
        Compute a positional SPE $(\sigma_i, \sigma_{-i})$ of $\mathcal{G}_i = (V, V_i, E_{\text{old}}, \chi, \Omega_i)$
        $W_i := \{v \in V : \sigma_i$ is winning in $(\mathcal{G}_i, v)\}$
        $E_{\text{new}} := E_{\text{new}} \setminus \{(u,v) \in E : u \in V_i \cap W_i$ and $v \neq \sigma_i(u)\}$
    end for
until $E_{\text{new}} = E_{\text{old}}$
for each $i \in \Pi$ do
    Compute equilibrium strategy $\tau_i$ of player $i$
end for
output $(\tau_i)_{i\in\Pi}$
```

Obviously, the running time of the algorithm depends on the running time of the algorithm we use for computing a positional subgame perfect equilibrium of a two-player zero-sum parity game. The best known algorithm for this problem, which also computes the winning regions (i.e. the set of vertices from which each player has a winning strategy) of the game, is due to Jurdziński [6]. For a game with at most n vertices, m edges and $d \geq 2$ different priorities, Jurdziński's algorithm runs in time

$$O\left(dm\left(\frac{n}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor}\right) .$$

Note that each strategy τ_i can be implemented by a finite automaton of size $O(|\Pi|^2|V|)$. Thus we have the following theorem.

Theorem 7. *Computing a finite-state subgame perfect equilibrium of a multiplayer parity game with k players, n vertices, m edges and at most $d \geq 2$ priorities for each player can be done in time*

$$O\left(kdm^2\left(\frac{n}{\lfloor d/2 \rfloor}\right)^{\lfloor d/2 \rfloor} + k^3n^2\right) .$$

In particular, Theorem 7 says that we can compute a subgame perfect equilibrium of a multiplayer parity game in polynomial time for classes of games with a bounded number of priorities. Moreover, if there exists a polynomial-time algorithm for computing a positional subgame perfect equilibrium of a two-player zero-sum parity game then Algorithm 1 can be made to run in polynomial time as well. Hence the problem of computing a subgame perfect equilibrium in an arbitrary multiplayer parity game is computationally not much harder than the corresponding problem for two-player zero-sum parity games.

4 Complexity

One can easily construct games where Algorithm 1 computes an equilibrium with a payoff of $(0, \ldots, 0)$ although there is an equilibrium with a payoff of $(1, \ldots, 1)$. This is unsatisfactory because, if we think of verification, we want as many components as possible to fulfil their specification. Therefore it seems desirable to find an equilibrium with a maximal payoff (a *maximal subgame perfect equilibrium*). This "maximisation problem" naturally corresponds to the decision problem SPE defined as follows:[2]

> Given an ω-regular multiplayer game $(\mathcal{G}, v_0)$ played on a finite arena and thresholds $x, y \in \{0,1\}^k$, decide whether $(\mathcal{G}, v_0)$ has a subgame perfect equilibrium with a payoff $\geq x$ and $\leq y$.

Note that we can find the payoff of a maximal subgame perfect equilibrium with k queries to the decision problem if k is the number of players. To solve the problem SPE, we use a reduction to the problem of deciding whether a given tree automaton defines a nonempty tree language.

Theorem 8. *The problem of deciding given a multiplayer Rabin game $(\mathcal{G}, v_0)$ played on a finite arena and thresholds $x, y \in \{0,1\}^k$ whether $(\mathcal{G}, v_0)$ has a subgame perfect equilibrium with a payoff $\geq x$ and $\leq y$ is in* Exptime. *If the number of players and pairs is bounded, the problem is in* Ptime.

Proof (sketch). Without loss of generality, we can assume that $\mathcal{G}$ is *binary*, i.e. every vertex of $\mathcal{G}$ has at most two successors. Then we can arrange all plays of $(\mathcal{G}, v_0)$ in an infinite binary tree with labels from the vertex set V. Given a strategy profile $(\sigma_i)_{i \in \Pi}$ of $(\mathcal{G}, v_0)$, we enrich this tree with a second label component that takes the value 0 or 1 if the strategy profile prescribes going to the left or right successor, respectively.

The algorithm works as follows: We construct two *alternating parity tree automata*. The first one checks whether some arbitrary tree with labels from the alphabet $V \times \{0,1\}$ is indeed a tree originating from a strategy profile of $(\mathcal{G}, v_0)$, and the second one checks for a tree originating from a strategy profile $(\sigma_i)_{i \in \Pi}$ of $(\mathcal{G}, v_0)$ whether $(\sigma_i)_{i \in \Pi}$ is a subgame perfect equilibrium with a payoff in between the given thresholds. The first automaton is actually a nondeterministic

[2] Here $\leq$ denotes the product ordering on $\{0,1\}^k$, i.e. $x \leq y$ if $x_i \leq y_i$ for all i.

tree automaton with trivial acceptance (every run of the automaton is accepting) and has $O(|V|)$ states. The second automaton has $O(kd)$ states and $O(1)$ priorities where k is the number of players and d is the maximum number of pairs in a player's winning condition. An equivalent nondeterministic parity tree automaton has $2^{O(kd \log kd)}$ states and $O(kd)$ priorities [11]. Finally, we construct the product automaton of the first nondeterministic parity tree automaton with the one constructed from the alternating one. As the former automaton works with trivial acceptance, the construction is straightforward and leads to a nondeterministic parity tree automaton with $O(|V|) \cdot 2^{O(kd \log kd)}$ states and $O(kd)$ priorities. Obviously, the tree language defined by this automaton is nonempty if and only if $(\mathcal{G}, v_0)$ has a subgame perfect equilibrium with a payoff in between the given thresholds. By [4], nonemptiness for nondeterministic parity tree automata can be decided in time polynomial in the number of states and exponential in the number of priorities. □

As any ω-regular multiplayer game can be reduced to an equivalent multiplayer parity game (and thus also to a multiplayer Rabin game), Theorem 8 implies the decidability of SPE. For LTL games the reduction gives an algorithm running in doubly exponential time. As the problem of deciding the winner in a two-player zero-sum LTL game is already 2EXPTIME-complete [14], this is optimal.

Corollary 9. *The problem* SPE *is decidable. For multiplayer LTL games the problem is* 2EXPTIME-*complete.*

We point out another consequence of our reduction. By Rabin's basis theorem [15], every regular, nonempty tree language contains a *regular tree*, i.e. a tree with only finitely many nonisomorphic subtrees. It is easy to see that a tree $t : \{0,1\}^* \to V \times \{0,1\}$ originating from a strategy profile $(\sigma_i)_{i \in \Pi}$ is regular if and only if each σ_i is a finite-state strategy. Thus we have the following theorem.

Theorem 10. *Let $(\mathcal{G}, v_0)$ be an ω-regular multiplayer game played on a finite arena and $x \in \{0,1\}^k$. Then $(\mathcal{G}, v_0)$ has a subgame perfect equilibrium with payoff x if and only if $(\mathcal{G}, v_0)$ has a finite-state subgame perfect equilibrium with payoff x.*

Intuitively, the theorem says that any subgame perfect equilibrium of an ω-regular multiplayer game played on a finite arena can be implemented by finite-state strategies. This is the best one can hope for because an arbitrary Nash or subgame perfect equilibrium, in general, cannot be implemented by positional strategies.

Example 2. Consider the following Büchi game with two players 1 and 2 played on the arena depicted in Fig. 2: Every vertex is controlled by player 1, and the winning condition of player i is to visit vertex i infinitely often. Obviously, player 1's finite-state strategy of alternating between visits to vertex 1 and vertex 2 induces a subgame perfect equilibrium of the game with payoff $(1,1)$. However, for any positional strategy of player 1 only one player wins the resulting play.

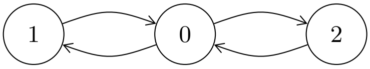


Fig. 2. A 2-player Büchi game

5 Definability

Let us now study the following question: Given winning conditions $W_1, \ldots, W_k \subseteq C^\omega$ defined with respect to a (finite) set C of colours and payoff thresholds x, y, in which logic can we define the class of initialised game arenas that, when equipped with the given winning conditions, admit a subgame perfect equilibrium with a payoff in between the given thresholds? Note that any game arena $G = (V, (V_i)_{i \in \Pi}, E, \chi)$, where $\chi : V \to C$, can be identified with the Kripke structure $(V, (V_i)_{i \in \Pi}, E, (P_c)_{c \in C})$ defined by $P_c = \{v \in V : \chi(v) = c\}$. Hence any logic that has a semantics for pointed Kripke structures can be used to define a class of initialised game arenas, one possible candidate being the *modal μ-calculus* L_μ, i.e. basic modal logic extended by least and greatest fixed points. Indeed, it is well known that for any fixed number of priorities the class of initialised two-player game arenas that admit a winning strategy for the first player in the corresponding two-player zero-sum parity game is L_μ-definable [3].

If G is a k-player game arena with colours in C and $W_1, \ldots, W_k \subseteq C^\omega$ are winning conditions, we write $G[W_1, \ldots, W_k]$ for the corresponding game. Then our result can be stated as follows.

Theorem 11. *Let $W_1, \ldots, W_k \subseteq C^\omega$ be ω-regular and $x, y \in \{0,1\}^k$. Then there exists a formula $\psi \in \mathrm{L}_\mu$ such that the following equivalence holds for every k-player game arena G with colours in C and every vertex v of G:*

$$G, v \models \psi \Leftrightarrow (G[W_1, \ldots, W_k], v) \text{ has a SPE with a payoff} \geq x \text{ and} \leq y \ .$$

Note that Theorem 11 is closely related to Theorem 8 and Corollary 9. Indeed, it can be shown that the formula ψ as defined in Theorem 11 can be constructed effectively. As the model-checking problem for L_μ on finite Kripke structures is decidable, this gives another method to solve the problem SPE.

As a special case, Theorem 11 implies that for every fixed ω-regular winning condition the class of initialised two-player game arenas that admit a winning strategy for player 0 in the corresponding two-player zero-sum game is definable in L_μ, a fact that, surprisingly, seems not to have been stated anywhere before.

Acknowledgements. This paper is largely based on the author's master's thesis [18]. I am grateful to Erich Grädel for his advice on this research. Parts of this work were done when the author was a visiting student at the University of Edinburgh. Thanks to Julian Bradfield and Kousha Etessami for their valuable comments on earlier versions of this paper.

References

1. K. Chatterjee, T. A. Henzinger and M. Jurdziński. Games with secure equilibria. In *Proceedings of the 19th Annual Symposium on Logic in Computer Science, LICS '04*, pages 160–169. IEEE Computer Society Press, 2004.
2. K. Chatterjee, M. Jurdziński and R. Majumdar. On Nash equilibria in stochastic games. In *Proceedings of the 13th Annual Conference of the Europan Association for Computer Science Logic, CSL '04*, volume 3210 of *LNCS*, pages 26–40. Springer-Verlag, 2004.
3. E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy (extended abstract). In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, FoCS '91*, pages 368–377. IEEE Computer Society Press, 1991.
4. E. A. Emerson, C. S. Jutla and A. P. Sistla. On model-checking for fragments of μ-calculus. In *Proceedings of the 5th International Conference on Computer Aided Verification, CAV '93*, volume 697 of *LNCS*, pages 385–396. Springer-Verlag, 1993.
5. D. Gale and F. M. Stewart. Infinite games with perfect information. In *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematical Studies*, pages 245–266. Princeton University Press, 1953.
6. M. Jurdziński. Small progress measures for solving parity games. In *Proceedings of the 17th Annual Symposium on Theoretical Aspects of Computer Science, STACS 2000*, volume 1770 of *LNCS*, pages 290–301. Springer-Verlag, 2000.
7. H. W. Kuhn. Extensive Games and the Problem of Information. In *Contributions to the Theory of Games II*, volume 28 of *Annals of Mathematical Studies*, pages 193–216. Princeton University Press, 1953.
8. D. A. Martin. Borel determinacy. *Annals of Mathematics*, 102:363–371, 1975.
9. D. A. Martin. An extension of Borel determinacy. *Annals of Pure and Applied Logic*, 49(3):279–293, 1990.
10. A. W. Mostowski. Games with forbidden positions. Technical Report 78, Instytut Matematyki, Uniwersytet Gdański, Poland, 1991.
11. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141(1–2):69–107, 1995.
12. J. F. Nash Jr. Equilibrium points in N-person games. *Proceedings of the National Academy of Sciences of the United States of America*, 36:48–49, 1950.
13. M. J. Osborne and A. Rubinstein. *A Course in Game Theory.* MIT Press, 1994.
14. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of the 31st Annual Symposium on Foundations of Computer Science, FoCS '90*, pages 746–757. IEEE Computer Society Press, 1990.
15. M. O. Rabin. Automata on infinite objects and Church's problem. *American Mathematical Society*, 1972.
16. R. Selten. Spieltheoretische Behandlung eines Oligopolmodells mit Nachfrageträgheit. *Zeitschrift für die gesamte Staatswissenschaft*, 121:301–324 and 667–689, 1965.
17. W. Thomas. On the synthesis of strategies in infinite games. In *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science, STACS '95*, volume 900 of *LNCS*, pages 1–13. Springer-Verlag, 1995.
18. M. Ummels. Rational behaviour and strategy construction in infinite multiplayer games. Master's thesis, RWTH Aachen, Germany, 2005.
19. W. Zielonka. Infinite games on finitely coloured graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1–2):135–183, 1998.

The Anatomy of Innocence Revisited

Russ Harmer and Olivier Laurent

PPS, CNRS and Université Paris 7

Abstract. We refine previous analyses of Hyland-Ong game semantics and its relation to λ- and $\lambda\mu$-calculi and present improved factorization results for bracketing and rigidity that can be combined in any order.

1 Introduction

Innocent strategies [4,7,2] provide models of (idealized programming languages based on) the λ- and $\lambda\mu$-calculi, the difference between these two calculi being expressed by the *bracketing condition*: whenever a strategy plays an *answer*, this must respond to the "pending" *question*, *i.e.* the most recent, as yet untreated, request. In [5], Laird analysed this situation and showed that an arbitrary innocent strategy σ can be decomposed as a well-bracketed innocent strategy $\mathcal{B}(\sigma)$ with access to an innocent but non-well-bracketed oracle `call/cc`. This semantic *factorization* mirrors the well-known result from proof theory that classical deductions can be rewritten as intuitionistic deductions with a few copies of Peirce's law as additional hypotheses. In [1], Danos & Harmer introduced a new constraint of *rigidity*, in a certain sense dual to bracketing, which restricts the use of `case` much as bracketing restricts the use of `call/cc` and showed that σ can be decomposed as a rigid innocent strategy $\overrightarrow{\mathcal{R}}(\sigma)$ with access to a non-rigid oracle `case`.

So we have a decomposition of the CCC $\mathbf{I}$ of innocent strategies into a "diamond" of subCCCs:

$$\begin{tikzcd}
& \mathbf{I} \arrow[dl, dash] \arrow[dr, dash] & \\
\mathcal{B} \arrow[dr, dash] & & \overrightarrow{\mathcal{R}} \arrow[dl, dash] \\
& \mathcal{B}\overrightarrow{\mathcal{R}} &
\end{tikzcd}$$

In $\mathcal{B}$, we can model `case` but not `call/cc` whereas in $\overrightarrow{\mathcal{R}}$ we can model `call/cc` but not `case`. However, neither factorization *preserves* the other constraint, *i.e.* eliminating `call/cc` reintroduces `case` and eliminating `case` reintroduces `call/cc`:

$$\begin{tikzcd}
& \mathbf{I} \arrow[dl] & \\
\mathcal{B} & & \overrightarrow{\mathcal{R}} \arrow[ll] \\
& \mathcal{B}\overrightarrow{\mathcal{R}} &
\end{tikzcd}
\qquad
\begin{tikzcd}
& \mathbf{I} \arrow[dr] & \\
\mathcal{B} \arrow[rr] & & \overrightarrow{\mathcal{R}} \\
& \mathcal{B}\overrightarrow{\mathcal{R}} &
\end{tikzcd}$$

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 224–235, 2006.

In this paper, we continue this analysis of innocent strategies with the aim of better understanding the role of answers in game semantics. To this end, we introduce an additional constraint, *backward rigidity* or *B-rigidity*, which extends the above decomposition to a "cube" of subCCCs (§3.3):

$$\begin{tikzcd}
 & \overleftarrow{\mathcal{R}} \arrow[rr, dash] \arrow[dl, dash] \arrow[dd, dash] & & \mathbf{I} \arrow[dl, dash] \arrow[dd, dash] \\
\mathcal{B}\overleftarrow{\mathcal{R}} \arrow[rr, dash] \arrow[dd, dash] & & \mathcal{B} \arrow[dd, dash] & \\
 & \mathcal{R} \arrow[rr, dash] \arrow[dl, dash] & & \overrightarrow{\mathcal{R}} \arrow[dl, dash] \\
\mathcal{B}\mathcal{R} \arrow[rr, dash] & & \mathcal{B}\overrightarrow{\mathcal{R}} &
\end{tikzcd}$$

This constraint can be seen as a dual to rigidity—which, henceforth, we rename as *forward rigidity*, reserving the term *rigidity* ($\mathcal{R}$) for the conjunction of the two—in that forward rigidity restricts the *elimination* (in the sense of natural deduction) of base type constants whereas backward rigidity restricts their *introduction*. The cube provides us with a taxonomy of logics and programming languages based on λ- and $\lambda\mu$-calculi: each node corresponds, via definability and full completeness theorems, to a fragment of μPCF (§3.4).

We then present (§4) a factorization for B-rigidity and modified factorizations for F-rigidity and bracketing, each of which preserves the other two constraints:

$$\begin{tikzcd}
 & \overleftarrow{\mathcal{R}} & \mathbf{I} \arrow[l] \\
\mathcal{B}\overleftarrow{\mathcal{R}} & \mathcal{B} \arrow[l] & \\
 & \mathcal{R} & \overrightarrow{\mathcal{R}} \arrow[l] \\
\mathcal{B}\mathcal{R} & \mathcal{B}\overrightarrow{\mathcal{R}} \arrow[l] &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \overleftarrow{\mathcal{R}} \arrow[dd] & & \mathbf{I} \arrow[dd] \\
\mathcal{B}\overleftarrow{\mathcal{R}} \arrow[dd] & & \mathcal{B} \arrow[dd] & \\
 & \mathcal{R} & & \overrightarrow{\mathcal{R}} \\
\mathcal{B}\mathcal{R} & & \mathcal{B}\overrightarrow{\mathcal{R}} &
\end{tikzcd}
\qquad
\begin{tikzcd}
 & \overleftarrow{\mathcal{R}} \arrow[dl] & & \mathbf{I} \arrow[dl] \\
\mathcal{B}\overleftarrow{\mathcal{R}} & & \mathcal{B} & \\
 & \mathcal{R} \arrow[dl] & & \overrightarrow{\mathcal{R}} \arrow[dl] \\
\mathcal{B}\mathcal{R} & & \mathcal{B}\overrightarrow{\mathcal{R}} &
\end{tikzcd}$$

This allows us to "navigate" (from $\mathbf{I}$) in the cube, applying factorizations in whichever order we like, and still being sure to end up in $\mathcal{BR}$. From a syntactic point of view, this explains how we can move from one language of the cube to another: a "smaller" language plus an appropriate oracle equals a "larger" language (*e.g.* λ-calculus plus `call/cc` equals $\lambda\mu$-calculus).

We conclude (§4.4) by examining the unary case where the factorizations can be simplified and the connection to logic becomes especially clear: $\mathcal{BR}$ = λ-calculus and $\mathcal{R}$ = $\lambda\mu$-calculus.

2 Innocent Game Semantics

This section briefly presents the definitions necessary to construct the category **I** of innocent strategies. A more detailed development can be found in [2].

2.1 Arenas and Plays

An ***arena*** A is a tuple $\langle M_A, \lambda_A, I_A, \vdash_A \rangle$ where

- M_A is a countable set of ***tokens***.
- $\lambda_A : M_A \to \{\mathsf{O},\mathsf{P}\} \times \{\mathsf{Q},\mathsf{A}\}$ ***labels*** each $m \in M_A$ as belonging to Opponent or to Player and as a Question or an Answer.
- I_A is a subset of $\lambda_A^{-1}(\mathsf{OQ})$ known as the ***initial moves*** of A.
- $\vdash_A$ is a binary ***enabling*** relation on M_A satisfying
  (e1) if $m \vdash_A n$ then $\lambda_A^{\mathsf{OP}}(m) \neq \lambda_A^{\mathsf{OP}}(n)$ and $n \notin I_A$;
  (e2) if $m \vdash_A n$ where $\lambda_A^{\mathsf{QA}}(n) = \mathsf{A}$ then $\lambda_A^{\mathsf{QA}}(m) = \mathsf{Q}$.

An arena where answers *never* enable other moves is called an A-***terminal*** arena. A ***flat*** arena has a single OQ-move and a set of PA-moves, all enabled by the O-move. For example, **bool** has an OQ, q, and two PAs, tt and ff, where $\mathsf{q} \vdash_{\mathbf{bool}} \mathsf{tt}$ and $\mathsf{q} \vdash_{\mathbf{bool}} \mathsf{ff}$. We similarly define $\bot$, **com** and **nat** as the flat arenas over $\varnothing$, $\{\mathsf{t}\}$ and $\{0, 1, 2, \ldots\}$ respectively. Note that a flat arena is always A-terminal.

A ***play*** in arena A is a string s over alphabet M_A with pointers between its occurrences such that, if s_i (the ith symbol of s) points to s_j then $j < i$, if s_j points to s_i then $s_i \vdash_A s_j$ and if s_i has no pointer then $s_i \in I_A$. We write $|s|$ for the length of s. A ***legal play*** in arena A is a play in A that also satisfies OP-***alternation***: $\lambda_A^{\mathsf{OP}}(s_i) \neq \lambda_A^{\mathsf{OP}}(s_{i+1})$ for $1 \leq i < |s|$. Each occurrence in a legal play s is an element m of M_A together with its pointer (unless $m \in I_A$); we call m plus its pointer a ***move*** of s. If m points to n in s, we say that n ***justifies*** m in s. We write $\mathcal{L}_A$ for the set of all legal plays in A.

The ***prefix*** ordering on strings extends to $\mathcal{L}_A$ with least element ε, the empty play. For $s, t \in \mathcal{L}_A$, we write $s \sqsubseteq t$ (resp. $s \sqsubseteq^{\mathsf{O}} t$, resp. $s \sqsubseteq^{\mathsf{P}} t$) when s is a (resp. O-ending, resp. P-ending) prefix of t. We fix the convention that $\varepsilon \sqsubseteq^{\mathsf{P}} s$ for any $s \in \mathcal{L}_A$. We write $s \wedge t$ for the ***longest common prefix*** of s and t, $\mathsf{ip}(s)$ or s^- for the ***immediate prefix*** of non-empty s and, provided the ***last move*** of s, written s_ω, has a pointer, $\mathsf{jp}(s)$ for the ***justifying prefix*** of s, *i.e.* that prefix of s ending with the move that justifies s_ω. We define $\mathsf{ie}(s) = \{t \in \mathcal{L}_A \mid \mathsf{ip}(t) = s\}$, the set of ***immediate extensions*** of s and, if $s \in \mathcal{L}_A$ and $m \in M_A$ such that s_ω enables m in A, we write $s \cdot m$ for the legal play obtained by adding m to the end of s, pointing to the last move.

We have the standard [6] constructors on arenas: the ***product*** $A \times B$ (and its infinite version A^ω), the ***par*** $A \parr B$ and the ***lift*** $\downarrow A$ from which we recover the familiar ***arrow*** $A \Rightarrow B$ as $(\downarrow A) \parr B$. If A and B are ***pointed*** arenas [only one initial move] then $A \parr B$ is also pointed and, in this special case, is written $A \oplus B$. All constructors preserve the property of being A-terminal.

2.2 The Ambient SMCC

A ***strategy*** σ for an arena A, written $\sigma : A$, is a non-empty set of P-ending legal plays of A which satisfies

- *prefix-closure*: if $s \in \sigma$ and $s' \sqsubseteq^{\mathsf{P}} s$ then $s' \in \sigma$;
- *determinism*: if $s \in \sigma$ and $t \in \sigma$ then $s \wedge t \in \sigma$.

The second condition amounts to asking for $s \wedge t$ to end with a P-move; so only Opponent can branch nondeterministically. We write $\mathsf{dom}(\sigma)$ for the ***domain*** of σ defined as $\bigcup_{s \in \sigma} \mathsf{ie}(s)$, all the O-ending plays of A *accessible* to σ.

We compose strategies $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ by parallel composition plus hiding, *i.e.* σ and τ synchronize on B and hide this from "the outside world", yielding $\sigma\,;\tau : A \Rightarrow C$. It can be shown that, by taking arenas as objects and strategies for $A \Rightarrow B$ as arrows between A and B, this notion of composition gives rise to an SMCC **G** [2].

2.3 The CCC of Innocent Strategies

We define the P-***view***, noted $\ulcorner s \urcorner$, of a non-empty legal play $s \in \mathcal{L}_A$ in two stages. First we extract a subsequence of s with pointers defined only for O-moves:

- $\ulcorner s \urcorner = s_\omega$, if s_ω is an initial move;
- $\ulcorner s \urcorner = \ulcorner \mathsf{jp}(s) \urcorner \cdot s_\omega$, if s_ω is a non-initial O-move;
- $\ulcorner s \urcorner = \ulcorner \mathsf{ip}(s) \urcorner\ s_\omega$, if s_ω is a P-move.

In words, we trace back from the end of s, following pointers from O-moves, excising all moves under such pointers, and "stepping over" P-moves, until we reach an initial move. In general, a P-move can "lose its pointer" (if it points to a move that gets erased in this way). The second stage of the definition specifies that, in such a case, the P-move has *no justifier* in the P-view (and so $\ulcorner s \urcorner \notin \mathcal{L}_A$); otherwise it keeps the same justifier as in s.

We say that a legal play $s \in \mathcal{L}_A$ satisfies P-***visibility*** iff $\ulcorner s \urcorner \in \mathcal{L}_A$. In words, no P-move of $\ulcorner s \urcorner$ loses its pointer. Note that this doesn't prevent a P-move of $\ulcorner t \urcorner$ losing its pointer, for t some proper prefix of s. We lift the definition of P-visibility to strategies in the obvious way: σ satisfies P-***visibility*** iff all $s \in \sigma$ do. Note that, for s in P-vis σ as opposed to arbitrary P-vis s, all $t \sqsubseteq^{\mathsf{P}} s$ do in fact satisfy P-visibility—since σ is closed under P-ending prefixes—so $\ulcorner t \urcorner \in \mathcal{L}_A$ for all the P-prefixes t of s.

If $s, t \in \mathcal{L}_A$ where s ends with a P-move, satisfies P-vis and $\ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner$ then we denote by $\mathsf{match}(s, t)$ the unique extension of t satisfying $\ulcorner s \urcorner = \ulcorner \mathsf{match}(s, t) \urcorner$, *i.e.* add the last move of s to t using the "same" pointer as in s. We can do this since, by assumption, the last move of s points in $\ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner$.

We now say that a *deterministic* P-vis strategy σ is ***innocent*** iff

$$s \in \sigma \wedge t \in \mathsf{dom}(\sigma) \wedge \ulcorner \mathsf{ip}(s) \urcorner = \ulcorner t \urcorner \Rightarrow \mathsf{match}(s, t) \in \sigma.$$

So an innocent strategy is completely determined by its ***view function*** $\ulcorner \sigma \urcorner$ defined to be $\{\ulcorner s \urcorner \mid s \in \sigma\}$. It can be shown that innocent strategies are closed under composition and form a subcategory **I** of **G** where the monoidal structure becomes Cartesian, *i.e.* **I** is a CCC. In the rest of this paper, we restrict to the full subCCC of **I** consisting of A-terminal arenas only.

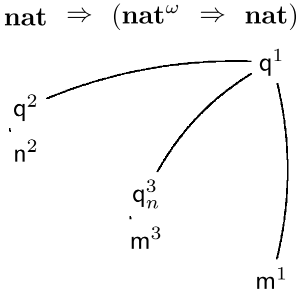
3 Bracketing and Rigidity

3.1 Backward Rigidity

An innocent strategy satisfies ***backward*** (or ***B-****)**rigidity*** iff every time it plays an answer, the *preceding* O-move was also an answer. This rules out strategies like $\mathsf{skip} = \{\varepsilon, \mathsf{q} \cdot \mathsf{t}\}$: **com** where Player produces an answer "from thin air".

3.2 Forward Rigidity

An obvious "dual" to B-rigidity applies the same condition to questions: every time the strategy plays a question, the preceding move must itself have been a question. We call such strategies ***forward*** (or ***F-****)**rigid***. In the setting of A-terminal arenas, this is equivalent to the notion of *rigid* strategy defined in [1]. This condition typically rules out case:



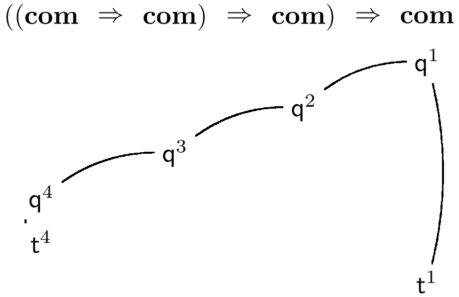
3.3 The Bracketing Condition

The P-view of an O-ending legal play s has generic form

$$\mathsf{OQ}((\mathsf{PQ} \curvearrowleft \mathsf{OQ})^*(\mathsf{PQ} \curvearrowleft \mathsf{OA})^*)^*$$

(where we've omitted Player's pointers for clarity). The rightmost OQ of $\ulcorner s \urcorner$ is called the ***pending question*** of s. An innocent strategy satisfies the ***bracketing*** condition iff, every time the strategy plays an answer, that answer is *justified* by the pending question.

This rules out strategies like call/cc à la Peirce:



3.4 The Cube of Subcategories

Each of the above constraints is preserved by composition, independently of the others. For this reason, we say that the constraints are *orthogonal.* As an immediate consequence of this, we can "unfold" the CCC of innocent strategies **I** into a cube of subcategories:

$$\begin{tikzcd}
 & \overleftarrow{\mathcal{R}} \arrow[rr, dash] \arrow[dl, dash] \arrow[dd, dash] & & \mathbf{I} \arrow[dl, dash] \arrow[dd, dash] \\
\mathcal{B}\overleftarrow{\mathcal{R}} \arrow[rr, dash] \arrow[dd, dash] & & \mathcal{B} \arrow[dd, dash] & \\
 & \mathcal{R} \arrow[rr, dash] \arrow[dl, dash] & & \overrightarrow{\mathcal{R}} \arrow[dl, dash] \\
\mathcal{B}\mathcal{R} \arrow[rr, dash] & & \mathcal{B}\overrightarrow{\mathcal{R}} &
\end{tikzcd}$$

As shown in [1,3], any innocent strategy with finite view function for (the arena interpreting) a simple type over a collection of flat arenas is denoted by a term in the following "Böhm tree" syntax (with appropriate typing rules) where Ω is a divergent term (or constant) of base type and k ranges over the (other) constants of base type:

$$\begin{aligned}
E &::= \Omega \mid [\alpha]\mathtt{k} \mid (\mathtt{case}\ (x)\vec{F}\ \ \overrightarrow{\mathtt{k} \mapsto E}) \\
F &::= \lambda\vec{x}\mu\alpha(E)
\end{aligned}$$

We can "unfold" this rather compact syntax into the following grammar:

$$\begin{aligned}
V &::= \Omega \mid [\alpha]\mathtt{k} \\
C &::= \Omega \mid (\mathtt{case}\ (x)\vec{F}\ \vec{M}) \\
E &::= V \mid C \\
M &::= \mathtt{k} \mapsto E \\
F &::= \lambda\vec{x}\mu\alpha(E)
\end{aligned}$$

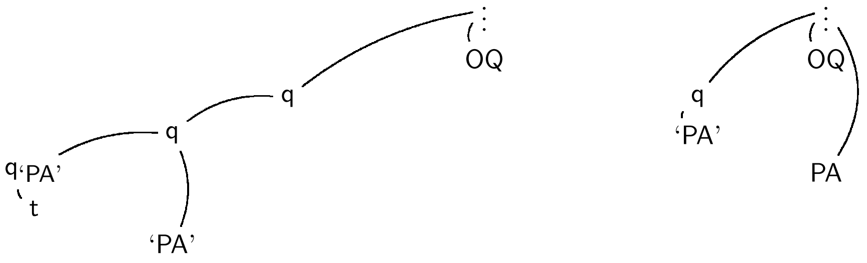
This more accurately reflects the game semantics in that each syntactic class corresponds to a certain kind of move—V for Player answers, C for Player questions, M for Opponent answers and F for Opponent questions—and allows us to easily identify the fragments corresponding to our three semantic constraints: to impose the bracketing condition, we simply erase all $\mu\alpha$s and $[\alpha]$s; to impose F-rigidity, we restrict M by $M' ::= \mathtt{k} \mapsto V$ and to impose B-rigidity, we restrict F by $F' ::= \lambda\vec{x}\mu\alpha(C)$.

4 Factorizations

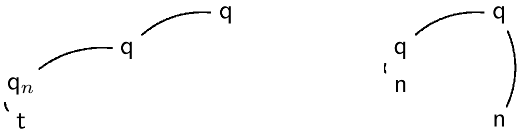
We now present factorizations, one for each of our constraints, each of which forces an innocent strategy to satisfy its constraint whilst preserving the other two. We fix, once and for all, an encoding of the answers (in a given arena) as natural numbers $\mathsf{A} \mapsto$ 'A' and a second encoding of answer-natural number pairs as natural numbers $\mathsf{A}, i \mapsto$ 'A_i'.

4.1 Backward Rigidity

To eliminate a violation of B-rigidity—a Player answer preceded by an Opponent question—we transform all OQ PA-ending P-views of σ into two new P-views:



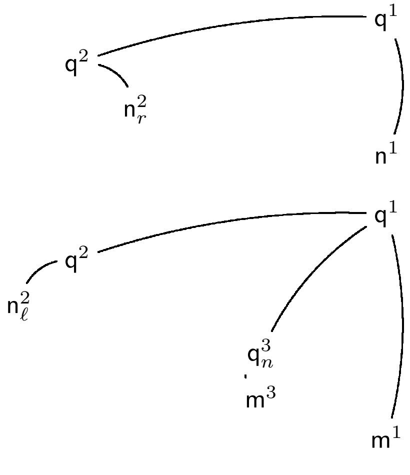
All other P-views remain unchanged. This determines a B-rigid innocent strategy $\overleftarrow{\mathcal{R}}(\sigma) : ((\mathbf{com}^{\omega} \Rightarrow \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow A$—which is well-bracketed and/or F-rigid if σ is—where we can recover σ by composing with const : $(\mathbf{com}^{\omega} \Rightarrow \mathbf{nat}) \Rightarrow \mathbf{nat}$, our *oracle* strategy, defined by the following view function:



4.2 Forward Rigidity

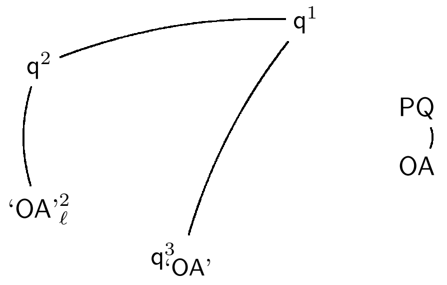
A violation of F-rigidity consists of an Opponent answer followed by a Player question. We would therefore like to transform the view function of $\sigma : A$ by "disguising" OAs as OQs so that σ can play all of its questions in an F-rigid manner. We can do this using $\mathsf{case}_{\oplus}$, our oracle for F-rigidity, with view function:

$$(\mathbf{nat} \;\oplus\; \mathbf{nat}) \;\Rightarrow\; (\mathbf{nat}^{\omega} \;\Rightarrow\; \mathbf{nat})$$

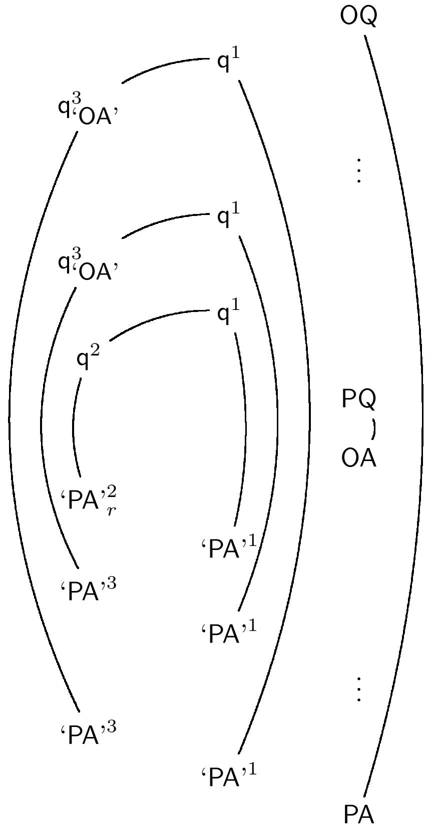


This strategy is deterministic, total, well-bracketed and B-rigid and we have an evident finite version $\mathsf{case}_{\oplus}^{k,l} : (\mathcal{F}_k \oplus \mathcal{F}_\ell) \Rightarrow (\mathcal{F}_\ell^k \Rightarrow \mathcal{F}_\ell)$ where $\mathcal{F}_k$ denotes a base type with k distinct values.

The factorization transforms PQ $\curvearrowleft$ OA arches of PQ-ending P-views of σ into PQ $\curvearrowleft$ OQ arches on $((\mathbf{nat} \oplus \mathbf{nat}) \Rightarrow (\mathbf{nat}^\omega \Rightarrow \mathbf{nat})) \Rightarrow A$:



For *well-bracketed* PA-ending P-views of σ, we must pop all the $\mathsf{q}_{\text{‘OA’}}$s introduced by the factorization, up to the pending question, so as to preserve bracketing:



The factorized strategy initiates popping by playing $\text{‘}\mathsf{PA}\text{’}^2_r$. The oracle propagates this directly to q^1. If the new pending question still belongs to the oracle, the strategy plays $\text{‘}\mathsf{PA}\text{’}^3_r$ and the oracle again propagates. This continues until we reach the pending question in A, whence PA is played. Note that this doesn't depend on σ at all: *all* factorized strategies will share the following essentially ***history free*** component, where Player always points to the pending question:

$$\begin{aligned}
\text{‘}\mathsf{PA}\text{’}^1 &\mapsto \mathsf{PA},\ \text{if the pending question is in } A\\
\text{‘}\mathsf{PA}\text{’}^1 &\mapsto \text{‘}\mathsf{PA}\text{’}^3,\ \text{otherwise}
\end{aligned}$$

To formally describe the σ-dependent component, we define, for $s \in \ulcorner\sigma\urcorner$, its (empty or singleton) ***principal*** P-view $\overline{s}$ and its set of ***auxiliary*** P-views $\mathcal{A}_s$:

$$\begin{aligned}
\varepsilon &\mapsto (\varepsilon, \varnothing)\\
s \cdot \mathsf{OQ}\,\mathsf{PQ} &\mapsto (\overline{s} \cdot \mathsf{OQ}\,\mathsf{q}^1 \cdot \mathsf{q}^2\,\mathsf{PQ}, \mathcal{A}_s)\\
s \cdot \mathsf{OA}\,\mathsf{PQ} &\mapsto (\overline{s}^{--} \cdot \mathsf{q}_{\text{‘}\mathsf{OA}\text{’}}\,\mathsf{q}^1 \cdot \mathsf{q}^2\,\mathsf{PQ}, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OA}\,\text{‘}\mathsf{OA}\text{’}^2_\ell\})\\
s \cdot \mathsf{OQ}\,\mathsf{PA} &\mapsto (\varnothing, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OQ}\,\mathsf{PA}\})\\
s \cdot \mathsf{OA}\,\mathsf{PA} &\mapsto (\varnothing, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OA}\,\mathsf{PA}\}),\ \text{if } s \cdot \mathsf{OA}\,\mathsf{PA} \text{ violates bracketing}\\
&\mapsto (\varnothing, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OA}\,\text{‘}\mathsf{PA}\text{’}^2_r\}),\ \text{otherwise}
\end{aligned}$$

Lemma 1. *If σ is an innocent strategy for A then*

$$\overline{\sigma} = \bigcup_{s \in \ulcorner\sigma\urcorner} \overline{s} \cup \mathcal{A}_s$$

is a view function for $((\mathbf{nat} \oplus \mathbf{nat}) \Rightarrow (\mathbf{nat}^\omega \Rightarrow \mathbf{nat})) \Rightarrow A$.

We can now formally define $\overrightarrow{\mathcal{R}}(\sigma)$ as the innocent strategy determined by $\overline{\sigma}$ and the history free component. This meshes perfectly with $\mathsf{case}_\oplus$ to implement our factorization:

Theorem 1. *If σ is an innocent strategy for A then $\overrightarrow{\mathcal{R}}(\sigma)$ is an F-rigid innocent strategy for $((\mathbf{nat} \oplus \mathbf{nat}) \Rightarrow (\mathbf{nat}^\omega \Rightarrow \mathbf{nat})) \Rightarrow A$ satisfying*

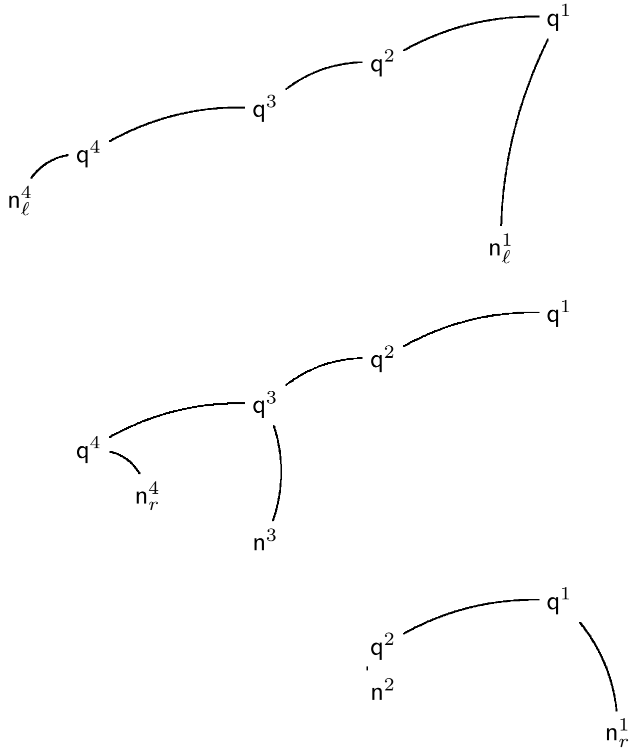
$$\mathsf{case}_\oplus\,;\overrightarrow{\mathcal{R}}(\sigma) = \sigma.$$

If σ is well-bracketed and/or B-rigid then so is $\overrightarrow{\mathcal{R}}(\sigma)$.

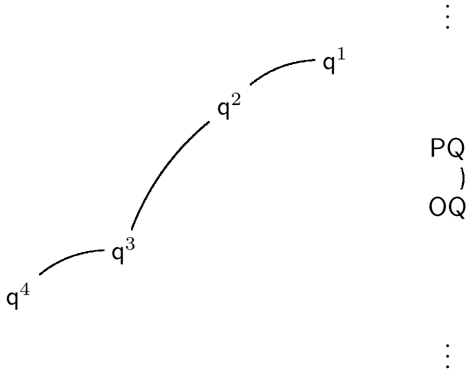
4.3 Bracketing

A violation of bracketing consists of a Player answer pointing beyond the pending question. We thus need to transform P-views in such a way that each $\mathsf{P} \curvearrowleft \mathsf{O}$ arch can, if necessary, be "popped" at a later point so as to recover a well-bracketed strategy. To do this, we use $\mathsf{call/cc}_\oplus$, a variant of $\mathsf{call/cc}$, as oracle:

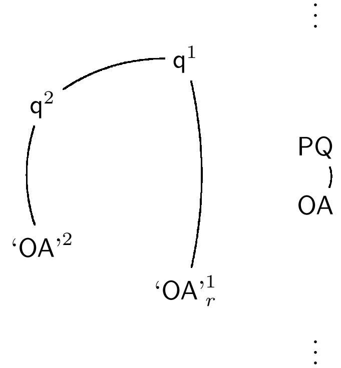
$$(((\mathbf{nat} \oplus \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow (\mathbf{nat} \oplus \mathbf{nat})$$



This strategy is deterministic, total and (B- and F-)rigid. If we restrict to finite enumerated types, we have $\mathsf{call/cc}_\oplus^{k,\ell} : (((\mathcal{F}_k \oplus \mathcal{F}_\ell) \Rightarrow \mathcal{F}_\ell) \Rightarrow \mathcal{F}_m) \Rightarrow (\mathcal{F}_k \oplus \mathcal{F}_m)$. Our factorization sandwiches each $\mathsf{PQ} \curvearrowleft \mathsf{OQ}$ arch of a P-view with $\mathsf{q}^1\,\mathsf{q}^2$ and $\mathsf{q}^3\,\mathsf{q}^4$. So, if we subsequently play ‘PA_j’$^4_\ell$, $\mathsf{call/cc}_\oplus$ replies with ‘PA_j’$^1_\ell$, “popping” the arch.



The factorization also transforms $\mathsf{PQ} \curvearrowleft \mathsf{OA}$ arches:



As for F-rigidity, the popping phase is mainly implemented by a history free component: the factorized strategy has only to initiate this process by providing an offset j—the number of OQs of σ between the answer we wish to play and its justifier. Formally, we map each P-view of $\ulcorner\sigma\urcorner$ on A to a principal P-view and set of auxiliary P-views on $((((\mathbf{nat} \oplus \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow (\mathbf{nat} \oplus \mathbf{nat})) \Rightarrow A$:

$$\begin{aligned}
\varepsilon &\mapsto (\varepsilon, \varnothing) \\
\mathsf{OQ}\,\mathsf{PQ} &\mapsto (\mathsf{OQ}\,\mathsf{q}^1 \cdot \mathsf{q}^2\,\mathsf{PQ}, \varnothing) \\
s \cdot \mathsf{OQ}\,\mathsf{PQ} &\mapsto (\overline{s} \cdot \mathsf{OQ}\,\mathsf{q}^3 \cdot \mathsf{q}^4\,\mathsf{q}^1 \cdot \mathsf{q}^2\,\mathsf{PQ}, \mathcal{A}_s) \\
s \cdot \mathsf{OA}\,\mathsf{PQ} &\mapsto (\overline{s}^{--} \cdot \text{‘}\mathsf{OA}\text{’}^1_r\,\mathsf{q}^1 \cdot \mathsf{q}^2\,\mathsf{PQ}, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OA}\,\text{‘}\mathsf{OA}\text{’}^2\}) \\
\mathsf{OQ}\,\mathsf{PA} &\mapsto (\mathsf{OQ}\,\mathsf{PA}, \varnothing) \\
s \cdot \mathsf{OA}\,\mathsf{PA} &\mapsto (\overline{s} \cdot \mathsf{OA}\,\mathsf{PA}, \mathcal{A}_s), \text{ if } s \text{ contains no non-initial } \mathsf{OQ}\text{s} \\
s \cdot \mathsf{OQ}\,\mathsf{PA} &\mapsto (\varnothing, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OQ}\,\mathsf{q}^3 \cdot \mathsf{q}^4\,\text{‘}\mathsf{PA}\text{’}^4_r\}), \text{ if } j = 0 \\
&\mapsto (\varnothing, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OQ}\,\mathsf{q}^3 \cdot \mathsf{q}^4\ \text{‘}\mathsf{PA}_{j-1}\text{’}^4_\ell\}), \text{ otherwise} \\
s \cdot \mathsf{OA}\,\mathsf{PA} &\mapsto (\varnothing, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OA}\,\text{‘}\mathsf{OA}\text{’}^2, \overline{s}^{--} \cdot \text{‘}\mathsf{OA}\text{’}^1_r\,\text{‘}\mathsf{PA}\text{’}^4_r\}), \text{ if } j = 0 \\
&\mapsto (\varnothing, \mathcal{A}_s \cup \{\overline{s} \cdot \mathsf{OA}\,\text{‘}\mathsf{OA}\text{’}^2, \overline{s}^{--} \cdot \text{‘}\mathsf{OA}\text{’}^1_r\,\text{‘}\mathsf{PA}_{j-1}\text{’}^4_\ell\}), \text{ otherwise}
\end{aligned}$$

Lemma 2. *If σ is an innocent strategy for A then*

$$\overline{\sigma} = \bigcup_{s \in \ulcorner\sigma\urcorner} \overline{s} \cup \mathcal{A}_s$$

is a view function for $((((\mathbf{nat} \oplus \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow (\mathbf{nat} \oplus \mathbf{nat})) \Rightarrow A$.

We write $\mathcal{B}(\sigma)$ for the innocent strategy determined by $\overline{\sigma}$ and the (almost) history free component:

$$\begin{aligned}
\text{‘}\mathsf{PA}_{j+1}\text{’}^1_\ell &\mapsto \text{‘}\mathsf{PA}_j\text{’}^4_\ell \\
\text{‘}\mathsf{PA}_0\text{’}^1_\ell &\mapsto \mathsf{PA}, \text{ if the pending question is initial} \\
&\mapsto \text{‘}\mathsf{PA}\text{’}^4_r, \text{ otherwise} \\
\text{‘}\mathsf{PA}\text{’}^3 &\mapsto \mathsf{PA}
\end{aligned}$$

Theorem 2. *If σ is an innocent strategy for A then $\mathcal{B}(\sigma)$ is a well-bracketed innocent strategy for $((((\mathbf{nat} \oplus \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow \mathbf{nat}) \Rightarrow (\mathbf{nat} \oplus \mathbf{nat})) \Rightarrow A$ satisfying*

$$\mathsf{call/cc}_{\oplus}\,; \mathcal{B}(\sigma) = \sigma.$$

If σ is F-rigid and/or B-rigid then so is $\mathcal{B}(\sigma)$.

4.4 The Unary Case

If we restrict ourselves to a single base type `com` with constants $\Omega, \mathtt{t} : \mathtt{com}$, we can simplify our factorizations and oracles. For B-rigidity, $\mathsf{skip} : \mathbf{com}$ suffices as oracle; the factorization simply inserts $\mathsf{q} \cdot \mathsf{t}$ just before all violating PAs. For F-rigidity, we use $\mathsf{seq} : \mathbf{com} \Rightarrow \mathbf{com} \Rightarrow \mathbf{com}$, the unary `case`, as oracle; the factorization simply transforms all $\mathsf{PQ} \curvearrowleft \mathsf{OA}$ arches into $\mathsf{q}^1 \curvearrowleft \mathsf{q}^3$, popping (if necessary) as usual.

For bracketing, Peirce's law $\mathtt{cc} : ((\mathbf{com} \Rightarrow \mathbf{com}) \Rightarrow \mathbf{com}) \Rightarrow \mathbf{com}$ acts as oracle. The factorization transforms every OQ PQ-ending P-view $s \in \ulcorner\sigma\urcorner$ by inserting $\mathsf{q}^1 \cdot \mathsf{q}^2\, \mathsf{q}^3 \cdot \mathsf{q}^4$ between the OQ and the PQ, where q^3 points to the q^2 occurring after the "answering justifier" of s: consider the unique, maximal P-view $t \in \ulcorner\sigma\urcorner$ extending s where all O-moves (after s) are answers; if t ends with an answer, the ***answering justifier*** is the question answered by the last move; otherwise (including the case where *no* such maximal P-view exists), the answering justifier is just the pending question of s. Syntactically, this corresponds to the translation $\mu\alpha(t) \mapsto (\mathtt{cc}\ \lambda\alpha(t))$ and $[\alpha]t \mapsto (\alpha)t$. Instead of popping arches one-by-one, this pops *everything* between a PA and its justifying question "in one go". This exploits the property (of the unary case) that we can statically determine the answering justifier of a P-view. In the general case, the answering justifier can only be known at runtime and so we have to pop arches dynamically.

References

1. V. Danos and R. Harmer. The anatomy of innocence. In *Proceedings, Tenth Annual Conference of the European Association for Computer Science Logic.* Springer Verlag, 2001.
2. R. Harmer. Innocent game semantics. Lecture notes, 2004–2006.
3. H. Herbelin. Games and weak-head reduction for classical PCF. In *Proceedings, Third International Conference on Typed Lambda Calculi and Applications*, Lecture Notes in Computer Science. Springer, 1997.
4. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.
5. J. Laird. Full abstraction for functional languages with control. In *Proceedings, Twelfth Annual IEEE Symposium on Logic in Computer Science.* IEEE Computer Society Press, 1997.
6. O. Laurent. Polarized games. *Annals of Pure and Applied Logic*, 130(1–3):79–123, December 2004.
7. H. Nickau. Hereditarily sequential functionals. In *Proceedings, Logical Foundations of Computer Science*, Lecture Notes in Computer Science. Springer-Verlag, 1994.

Testing Probabilistic Equivalence Through Reinforcement Learning

Josée Desharnais, François Laviolette,
and Sami Zhioua

IFT-GLO, Université Laval, Québec (QC) Canada, G1K 7P4
{firstname.lastname}@ift.ulaval.ca

Abstract. We propose a new approach to verification of probabilistic processes for which the model may not be available. We use a technique from Reinforcement Learning to approximate how far apart two processes are by solving a Markov Decision Process. If two processes are equivalent, the algorithm will return zero, otherwise it will provide a number and a test that witness the non equivalence. We suggest a new family of equivalences, called K-moment, for which it is possible to do so. The weakest, 1-moment equivalence, is trace-equivalence. The others are weaker than bisimulation but stronger than trace-equivalence.

1 Introduction

In program verification, the goal is typically to check automatically whether a system (program, physical device, protocol, etc.) conforms to its pre-established specification. For non-probabilistic systems, one usually expects equivalence between the two, and most of the time this equivalence is chosen to be bisimulation. In the verification of probabilistic systems it has been observed [8] that the comparison between the program and the specification should not be based on equivalences: one reason is that the probabilities involved often come from approximations of the actual numbers. Hence a slight difference in the probabilities between two processes should not necessarily be interpreted as non equivalence. Instead, one is interested in a notion of distance or divergence[1] that quantifies *how far apart* the processes are. When defining a divergence or distance, we have two focus: its computability of course but also the relation induced by zero distance. The actual value of the distance is usually not relevant but the derived relation, which may be for example bisimulation or trace equivalence, is a guide to evaluate the power or adequacy of the distance or divergence.

In real scenarios, the model of the implementation is rarely known and the available information can only be gathered by interacting with the system. Consequently, verification in this setting has to be based on some form of sampling (or testing). In their famous paper on probabilistic transition systems [13], Larsen and Skou have defined a test language that corresponds to *probabilistic bisimulation.* Two processes are bisimilar if and only if they accept the same tests

[1] We use the word divergence to mean a distance that may not satisfy the triangle inequality and symmetry.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 236–247, 2006.

with the same probabilities. From the maximal difference over the probabilities on these tests, Van Breugel et. al. [2] have defined a divergence (in fact a pseudo-metric) that quantifies the difference between the processes. However, the tests defined by Larsen and Skou have a copy construct that represents running many tests on a given state, and this recursively. The need to maintain an arbitrary number of replicas of states is an obstacle to automatisation and has been an argument against bisimulation which is thus considered too strong, even for non-probabilistic processes.

In this paper, we explore alternative equivalences and divergences that do not require to maintain an *unbounded* number of replicas. In particular, we suggest a new family of equivalences called K-moment that only need a *non-recursive* form of replication. We also propose an algorithm to compute these divergences using Reinforcement Learning (RL) methods; these are applicable even when the model is not available. While verification techniques can deal with processes of about 10^{12} states, RL algorithms do a lot better; for example, the *TD-Gammon* program deals with more than 10^{40} possible states [15]. The key idea of our approach is to define a Markov Decision Process (MDP) out of the processes to be tested and to interpret the optimal value of this MDP as a divergence between the processes. Moreover, the algorithm we propose outputs a test that witnesses the computed divergence. In [5], we showed how it can be done for trace-equivalence; we now extend the approach to a large family of testing equivalences.

The plan of the paper is as follows. In the following section, we point out the difficulties behind testing probabilistic bisimulation and define the K-moment equivalence. In Section 3, we informally expose our approach via a one player stochatic game and give the key definition of the MDP, the associated theorems, and the experimental results. Section 4 shows briefly how the approach can be applied to other equivalence notions.

2 Testing Equivalences

We consider probabilistic *reactive* systems where actions are meant to be synchronized through interaction with the environment and where no internal actions occur. Our models are *Labelled Markov Processes* (LMPs) [1]; while they can be uncountable in general, we restrict ourselves to countable ones. Finite LMPs are also called Probabilistic labelled transition systems or Markov decision processes without rewards. A countable LMP is a tuple (S, i, Act, P) where S is a countable set of processes, $\mathsf{i} \in S$ the initial process, Act a finite set of actions, and $P(s,a)$ a sub-probability distribution on S, for $s \in S$ and $a \in Act$ (see Fig. 1). We will write $\mathcal{S}$ generically for this tuple, and assume that subscripts propagate to the four elements. We use the notation $P_{sX}(a)$ for $P(s,a)(X)$, the probability that an a-transition from s ends in X. Given a trace τ (i.e., a sequence of actions), $P_{\mathsf{i}s}(\tau)$ is the probability to reach s with τ from the initial state i. We will always assume our models to be tree like; up to bisimulation [1], it is always possible.

Bisimulation

Larsen and Skou showed that probabilistic bisimulation can be characterized by a testing scenario [13]. Their test language has the following syntax:

$$\mathcal{T}_{LS}: \qquad t ::= \omega \mid a.t \mid (t_1, \ldots, t_n)$$

ω is a dummy test that always terminates with success; test $a.t$ consists in executing action a and, in case of success, proceeding with test t; finally, test $(t_1, \ldots, t_n)$ consists in making n copies of the current process and then executing test t_i on the i^{th} copy for $i = 1, \ldots, n$. The execution of a test may result in several possible observations. Let $a^{\checkmark}$ represents the success of action a and $a^{\times}$ its failure. The observation set of test t is recursively defined as follows :

$$O_\omega = \{\omega\}, \qquad O_{a.t} = \{a^{\times}\} \cup \{a^{\checkmark}.e \mid \ e \in O_t\}, \qquad O_{(t_1,\ldots,t_n)} = O_{t_1} \times \ldots \times O_{t_n}$$

To each test t is associated a probability distribution $Q_t^s(e)$ on O_t; it represents the probability to witness observation e after running t on process s and is defined as :

$$Q^s_\omega(\omega) = 1, \qquad Q^s_{(t_1,\ldots,t_n)}((e_1,\ldots,e_n)) = \textstyle\prod_i Q^s_{t_i}(e_i) \ \text{ where } e_i \in O_{t_i} \ \forall i,$$

$$Q^s_{a.t}(a^{\times}) = 1 - P_{sS}(a), \qquad Q^s_{a.t}(a^{\checkmark}.e) = \sum_{s' \in S} P_{ss'}(a) Q^{s'}_t(e) \ \text{ where } e \in O_t.$$

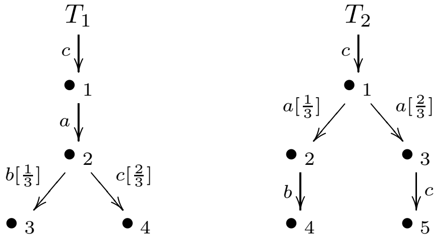
Theorem 1 ([13]). *Two processes are probabilistic bisimilar iff they yield the same probability distribution on observations for any test of the grammar $\mathcal{T}_{LS}$.*

The replication construct in the test language $\mathcal{T}_{LS}$ makes bisimulation very difficult to check in practice. Indeed, this construct requires to make n replicas of the current process and to execute a test on each replica. Since this construct is defined recursively on tests, there is no bound on the number of replicas that must be kept in memory. However, as is well known, bisimulation cannot be tested without recursive replication. In this work, we propose equivalences that make sure to limit the number of replicas[2]. This allows us to propose an algorithm that uses efficient techniques of RL to estimate the divergence between LMPs. A major advantage of this approach is that it runs even if the model of the implementation is unknown, contrarily to Van Breugel and Worrell's work [3]. In the latter, a pseudo metric that corresponds to bisimulation is computed through a linear programming algorithm.

***K*-moment Equivalences**

Trace-equivalence, contrarily to bisimulation, can be tested without the need to create replicas recursively, but for many applications, it does not discriminate enough. Especially, it cannot discriminate between the following two processes:

[2] In the classification of Van Glabbeek [9], the testing machine we assume is equipped with (1) a series of buttons (one for each action), (2) a reset button and (3) a replication button to generate copies of the current process. To avoid recursive replication, we delete copies in memory once a transition happens from one state to the next.



Of course bisimulation does discriminate between them, and therefore can catch the fact that the probabilistic choice happens at different levels in both processes. One may wonder if equivalences without recursive replication can catch this difference. Such equivalences, if they can be tested efficiently, should be a good compromise between the fact that bisimulation is very costly to test and the fact that trace-equivalence does not discriminate enough.

It is well known that probabilistic bisimulation can be formulated as the greatest fixed point of F defined as follows. Given an equivalence relation R, we say that two states s_1, s_2 are $F(R)$ equivalent if they have the same probability to jump to an equivalence class of R with every sequence of actions. More formally $\sum_{t\in C} P_{s_1 t}(\tau) = \sum_{t\in C} P_{s_2 t}(\tau)$ for all R-equivalence class C. Observe that trace-equivalence is simply $F(S\times S)$. Hence, in order to naturally define an equivalence whose discrimination power is between bisimulation and trace-equivalence, one can consider $\cap_{a\in Act} F(\sim_a)$ where $\sim_a$ is the equivalence that identifies states that have the same total probability to perform action a, that is $P_{sS}(a)$. This relation does discriminate T_1 and T_2 and any two trace-equivalent processes on which probabilistic choices happen at different levels. It can be tested without recursive replicas but, unfortunately, the number of needed replicas at a particular state is unbounded. However, we will see that it is the limit of a sequence of "bounded replicas" equivalences. First we need the following:

Definition 1. *Let (S, i, Act, P) be an LMP, $a \in Act$ and $\tau \in Act^*$. We define $X_{\tau,a} : S \cup \{Dead\} \to [0,1]$ the random variable representing the probability to perform action a after having run trace τ. Dead is the outcome of the experiment of failing to perform τ. Equivalently, for $p > 0$*

$$Pr(X_{\tau,a} = p) = \sum_{s:P_{sS}(a)=p} P_{\mathsf{i}s}(\tau)$$

and $\{X_{\tau,a} = 0\} = \{s : P_{sS}(a) = 0\} \cup \{Dead\}$.

At first sight, it is surprising to see a random variable taking probability values, but recall that we are performing tests on processes and are indeed observing the probabilities that these tests be accepted.

It is easy to see that two processes $\mathcal{S}_1$ and $\mathcal{S}_2$ are $\cap_{a\in Act} F(\sim_a)$-equivalent if, and only if, for any trace $\tau \in Act^*$ and action $a \in Act$, the random variables $X^{\mathcal{S}_1}_{\tau,a}$ and $X^{\mathcal{S}_2}_{\tau,a}$ are equal. Since any two *discrete* random variables are equal if and only if they have the same moments[3], the following is a natural relaxation of the preceding equivalence:

[3] Recall that, the i^{th} moment of a random variable X is the expected value of X^i.

Definition 2. *Let $K \in \mathbb{N}^*$. Two LMPs $\mathcal{S}$ and $\mathcal{S}'$ are K-moment equivalent, if and only if, $\forall\ \tau \in Act^*, \forall\ a \in Act\ \ X^{\mathcal{S}}_{\tau,a}$ and $X^{\mathcal{S}'}_{\tau,a}$ have exactly the same first K moments. That is, $E((X^{\mathcal{S}}_{\tau,a})^k) = E((X^{\mathcal{S}'}_{\tau,a})^k)$ for $k \leq K$, or equivalently, $\sum_{s\in S} P_{\mathsf{i}s}(\tau)\ (P_{sS}(a))^k = \sum_{s'\in S'} P'_{\mathsf{i}'s'}(\tau)\ (P'_{s'S'}(a))^k$ for $k \leq K$.*

It is easy to see that 1-moment equivalence corresponds to trace-equivalence. K-moment also has a nice characterization in term of tests:

Definition 3. *Let $K \in \mathbb{N}^*$. The K-moment test grammar is defined as:*

$$\mathcal{T}_{Kmoment}: \qquad t ::= \omega \mid a^k.t$$

with $k \leq K$such that (1) the test $a^k.t$ consists in running action a on k copies of the current process and if the last action succeeds, proceed with test t on the last copy (and delete the other copies);
(2) the observations corresponding to the test $a^k.t$ are given by:

$$O_{a^k.t} = \{a^{k^\times}\} \cup \{a^{k^{\not\checkmark}}.e \mid e \in O_t\} \cup \{a^{k^\checkmark}.e \mid e \in O_t\}$$

$a^{k^\times}$ is the observation that the last action has failed (and maybe others); $a^{k^{\not\checkmark}}$ means that an action failed but not the last one and finally $a^{k^\checkmark}$ is the observation that all copies succeeded on a. The probability distribution on observations is:

- $Q^s_{a^k.t}(a^{k^\times}) = 1 - P_{sS}(a)$
- $Q^s_{a^k.t}(a^{k^{\not\checkmark}}.e) = (1 - P_{sS}(a)^{k-1})\ \sum_{s'\in S} P_{ss'}(a) Q^{s'}_t(e)$
- $Q^s_{a^k.t}(a^{k^\checkmark}.e) = P_{sS}(a)^{k-1}\ \sum_{s'\in S} P_{ss'}(a) Q^{s'}_t(e)$.

For processes T_1 and T_2 defined above, the test $t = c.a.b^2.\omega$ yields different probability distributions on observations: $Q^{T_1}_t(c^\checkmark.a^\checkmark.b^{2^\checkmark}.\omega) = 1\ 1\ (\frac{1}{3})^2\ 1 = \frac{1}{9}$ while $Q^{T_2}_t(c^\checkmark.a^\checkmark.b^{2^\checkmark}.\omega) = 1\ \frac{1}{3}\ (1)^2\ 1 = \frac{1}{3}$. Note also that $Q^{T_1}_t(c^\checkmark.a^\checkmark.b^{2^{\not\checkmark}}.\omega) = 1\ 1\ (\frac{2}{3}\frac{1}{3})\ 1$, whereas $Q^{T_1}_t(c^\checkmark.a^\checkmark.b^{2^\times}) = 1\ 1\ (1 - \frac{1}{3})$.

Theorem 2. *Let $K \in \mathbb{N}^*$. Two LMPs are K-moment equivalent iff they yield the same probability distribution on observations of tests generated from $\mathcal{T}_{Kmoment}$.*

This equivalence is interesting in two ways: it closes the gap between bisimulation and trace-equivalence and it is testable without recursive replication. We will now take advantage of its testability.

3 Testing Without the Model

We want to define a divergence between "Spec", a model of the specification and "Impl", a real system; the model of the latter is not necessarily available but we can interact with it (as a black-box) via the testing machine. We also want this divergence to come as the solution of an MDP, the basic ingredient of RL techniques, on which the learning algorithm works. The MDP will be defined in Section 3.2. The rewards in the MDP have to be chosen carefully to make sure that the optimal value will indeed define a divergence as we expect: this is the subject of Section 3.1. We expose our approach in the form a one-player stochastic game, the player being the personification of the learning algorithm.

We illustrate our approach for K-moment equivalence but the ideas can be adapted for other equivalences testable through a recursive replication-free test grammar (see Section 4).

3.1 *K*-Moment Equivalences Through a Stochastic Game

When interacting with processes "Spec" and "Impl", the player's goal is to detect difference between the two. Hence the game (and its corresponding MDP) should give him low reward when they *behave the same* (i.e., both succeed or both fail) and high reward otherwise. However, the processes are probabilistic and hence a process may behave differently on different trials of the same action, which could lead the player to find a big difference between identical processes. This will happen more likely when the choice at a state is "wide", more technically, when the entropy is high. To compensate this uncertainty, we introduce a third process, called "Clone" which is simply a copy of the specification but given in the form of a black-box (exactly as "Impl") (see Figure 1). The player will get a high reward if "Impl" and "Spec" differ for some action, but this reward could be cancelled if "Spec" and "Clone" also do. Recall that the player does not see the states reached in "Impl" and "Clone" but does see what happens in "Spec".

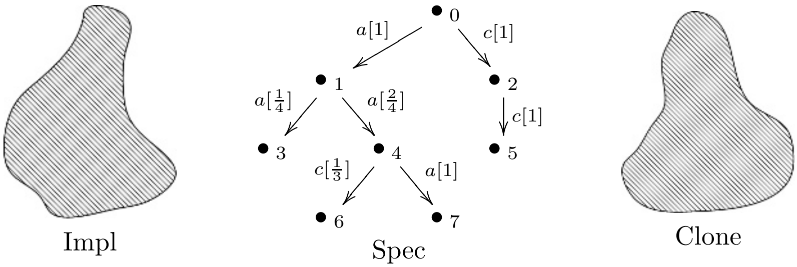


Fig. 1. Implementation, Specification, and Clone

Game$_{Kmoment}$**:** The three processes start in their initial states; then

Step 1: The player chooses an action a and an integer $k < K$, and makes a prediction `Pred` on its success or failure on "Spec" . We will denote the player choice by $a^{k^{\checkmark}}$ for success and by $a^{k^{\times}}$ otherwise.

Step 2: a^k is run on "Impl", "Spec" and "Clone" as in Definition 3. Let $(o_I, o_{Sp}, o_C) \in \{a^{k^{\times}}, a^{k^{\not\checkmark}}, a^{k^{\checkmark}}\}^3$ be the outcome of this experiment.

Step 3: Get reward as defined below. If the last occurrence of action a succeeds on the three processes, go to Step 1 with the three processes in their reached state. Else the game ends.

The player gets a reward according to the following formula:

$$R := \big(o_{Sp} \sim \texttt{Pred}\big)\big((o_I \not\sim o_{Sp})) - (o_{Sp} \not\sim o_C)\big)$$

where 0 and 1 are used as both truth values and numbers and the relation $\sim$ is a relaxation of equality to identify cases where a failure happens: i.e., $a^{k^{\times}} \sim a^{k^{\not\checkmark}} \not\sim a^{k^{\checkmark}}$.

For example, if $a^{k^{\checkmark}}$ is selected and the observation is $(a^{k\text{✗}}, a^{k^{\checkmark}}, a^{k^{\checkmark}})$ we obtain a reward of $(a^{k^{\checkmark}} \sim a^{k^{\checkmark}})((a^{k\text{✗}} \not\sim a^{k^{\checkmark}}) - (a^{k^{\checkmark}} \not\sim a^{k^{\checkmark}})) = 1\ (1-0) = 1$, but for $(a^{k\text{✗}}, a^{k^{\times}}, a^{k^{\checkmark}})$, we obtain $0\ (0-1) = 0$.

With the rewards so defined, we will show in Section 3.2 that "Spec" and "Impl" are K-moment equivalent if, and only if, the optimal strategy has expected reward zero. In other words, we will show that the optimal value of the MDP defined from this game yields a suitable notion of divergence.

Remark 1. It is not clear at first sight why the prediction is important in Step 1. It happens that by just running a^k on the processes and collecting the rewards we do not reach our goal totally (that is, without multiplying by $o_{Sp} \sim \texttt{Pred}$). If "Spec" and "Impl" are K-moment equivalent, the optimal strategy would indeed have expected reward zero, as wanted. However, the converse would not be true: there are K-moment inequivalent LMPs for which the optimal strategy would have expected reward zero. This is not what we want because they would get a divergence zero. Here is an example: for $K = 1$, consider three systems with one a-transition from one state to another one. The probability of this transition is $\frac{1}{2}$ in "Spec" (and "Clone") and 1 in "Impl". Note that such examples all exhibit a specific form of symmetry in "Spec".

Remark 2. This game has been inspired by a well known and well studied divergence, the Kullback-Leibler divergence. The idea is that two processes are "equivalent" via testing if, and only if, they yield the same probability distributions on observations for any test generated from the given test grammar. Hence, the divergence between two processes could be defined with the help of a divergence between the *probability distributions* on test observations. The Kullback-Leibler divergence (KL divergence) would be a candidate: it is defined, for two distributions Q and P, as $\mathrm{KL}(Q\|P) := \mathrm{E}_{h\sim Q} \ln \frac{1}{P(h)} - \mathrm{E}_{h\sim Q} \ln \frac{1}{Q(h)}$ [4]. Unfortunately, because of the high number of possible tests (on huge systems), the maximum value over all Kullback-Leibler divergences is not tractable. Nevertheless, let us describe the analogy between $\mathbf{Game}_{Kmoment}$ and the KL divergence. The entropy of P relativised by Q ($\mathrm{E}_{h\sim Q} \ln \frac{1}{P(h)}$ in the above formula) can be seen as how likely we can obtain different observations when interacting (via some test t) with both "Spec" and "Impl". On the other hand, the entropy of the distribution Q ($\mathrm{E}_{h\sim Q} \ln \frac{1}{Q(h)}$ in the above formula) can also be seen as a quantification over the likelihood to obtain different observations when running the same action on "Spec" twice. Thus, the game expresses the same kind of tradeoff as the KL divergence. This is a first indication that one can derive the notion of divergence we are looking for.

3.2 The Reinforcement Learning Framework

In artificial intelligence, Markov Decision Processes (MDPs) offer a popular mathematical tool for planning and learning in the presence of uncertainty [11]. MDPs are a standard formalism for describing multi-stage decision making in probabilistic environments (what we called a one-player stochastic games in the preceding section). The objective of the decision making is to maximize a cumulative measure of long-term performance, called the reward.

In an MDP, an agent interacts with a stochastic environment at a discrete, low-level time scale. On each time step t, the agent observes its current state $s_t \in S$ and chooses an action a_t from an action set A. One time step later, the agent transits to a new state s_{t+1}, and receives a reward r_{t+1}. For a given state s and action a, the expected value of the immediate reward is denoted by $R^a_{s\ S}$ and the transition to a new state s' has probability $Pr^{\mathcal{M}_\mathcal{K}}_{s\ s'}(a)$, regardless of the path taken by the agent before state s (this is the *Markov property*). The goal in solving MDPs is to find a *way of behaving*, or policy, which yields a maximal reward. Formally, a policy is defined as a probability distribution for picking actions in each state. For any policy $\pi : S \times A \rightarrow [0, 1]$ and any state $s \in S$, the *value function* of π for state s is defined as the expected infinite-horizon discounted return from s, given that the agent behaves according to π: $V^\pi(s) := E_\pi\{r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots | s_t = s\}$ where i is the initial state and γ is a factor between 0 and 1 used to discount future rewards. The objective is to find an optimal policy, π^* which maximizes the value $V^\pi(s)$ of each state s. The *optimal value function*, V^*, is the unique value function corresponding to any optimal policy.

If the MDP has finite state and action spaces, and if a model of the environment is known (i.e., state space S, immediate rewards $R^a_{s\ s'}$ and transition probabilities $Pr^{\mathcal{M}_\mathcal{K}}_{s\ s'}(a)$), then DP algorithms (namely policy evaluation [15]) can compute V^π for any policy π. Similar algorithms can be used to compute V^*. RL methods, in contrast, compute approximations to V^π and V^* directly based on the interaction with the environment, without requiring a complete model or finiteness of the MDP. Only the state space of the MDP and the knowledge of the exact current state at each step of the interaction are required. This is exactly what we are looking for.

Constructing the MDP. We now define the MDP (denoted $\mathcal{M}_K$) with which the divergence between two LMPs, "Impl" and "Spec", will be computed. Interacting with $\mathcal{M}_K$ will be like executing $\text{Game}_{Kmoment}$. The model of "Spec" is needed and must be in a tree-like representation (up to bisimulation, this is always possible). Since it has a tree-like structure, for any of its state s, there is a unique sequence of actions (or trace, denoted *tr.s*) from the initial state to s. For the LMP "Impl", only the knowledge of all possible conditional probabilities $P^I(a^{k^\checkmark}|\tau)$ and $P^I(a^{k^\times}|\tau)$ of observing the success or failure of an action a^k given any successfully executed trace τ is required; note that we define the trace corresponding to the execution of the i first steps of the test $a_1^{k_1^-} a_2^{k_2^-} \ldots a_n^{k_n^-}$ as $\tau = a_1 a_2 \ldots a_i$ (we write a^{k^-} for $a^{k^\checkmark}$ or $a^{k^\times}$) . We write $P^C(a^{k^\checkmark}|\tau)$ and $P^C(a^{k^\times}|\tau)$ for the same conditional probabilities but on a copy of the first LMP (called "Clone"): this is for readability and is no additional information since $P^C(a^\checkmark|\tau) = P^{Spec}(a^\checkmark|\tau)$.

The state space of the MDP $\mathcal{M}_K$ will be the state space of the LMP "Spec" plus one extra state, called *Dead*. This state corresponds in the MDP to the fact that $\text{Game}_{Kmoment}$ is over: i.e., given that a^{k^-} is the last choice of the player, then one of the three LMPs failed to execute the k^{th} action a. Any other state of $\mathcal{M}_K$ represents the current state of the LMP "Spec" during the execution of Game$_{Kmoment}$. The probability transitions ($Pr^{\mathcal{M}_\mathcal{K}}_{s\ s'}(a^{k^-})$) and the average rewards signals ($R^{a^{k^-}}_{s\ S}$) therefore follows from the rules of Game$_{Kmoment}$[4]. More formally:

Definition 4. Given "Impl", "Spec"$=$ ($States, \mathsf{i}, Actions, Pr^{Spec}$), and "Clone", the set of states of the MDP $\mathcal{M}_K$ is $S := States \cup \{Dead\}$, with initial state i; the action set is $Act := \{a^{k^{\checkmark}} | a \in Actions, 1 \leq k \leq K\} \cup \{a^{k^{\times}} | a \in Actions, 1 \leq k \leq K\}$. The next-state probability distribution is the same for $a^{k^{\checkmark}}$ and $a^{k^{\times}}$; it is defined below, followed by the definition of the reward function.

$$Pr^{\mathcal{M}_\mathcal{K}}_{s\ s'}(a^{k^-}) := \begin{cases} Pr^{Spec}_{s\ s'}(a)\ P^I(a^{\checkmark} | tr.s)\ P^C(a^{\checkmark} | tr.s) & \text{if } s' \neq Dead \\ 1 - P^{Spec}(a^{\checkmark} | s)\ P^I(a^{\checkmark} | tr.s)\ P^C(a^{\checkmark} | tr.s) & \text{if } s' = Dead \end{cases}$$

$$R^{a^{k^-}}_{s\ s'} := \begin{cases} P^{Spec}(a^{(k-1)^-} | s)\ \Delta^{a^{(k-1)^-}}_{tr.s} & \text{if } s' \neq Dead \text{ and } k > 1 \\ 0 & \text{if } s' \neq Dead \text{ and } k = 1 \end{cases}$$

$$R^{a^{k^-}}_{s\ Dead} := \frac{1}{Pr^{\mathcal{M}_\mathcal{K}}_{s\,Dead}(a^{k^-})} \left(P^{Spec}(a^{k^-} | s)\ \Delta^{a^{k^-}}_{\tau} - \sum_{s' \in S \setminus \{Dead\}} Pr^{\mathcal{M}_\mathcal{K}}_{s\ s'}(a^{k^-})\ R^{a^{k^-}}_{s\ s'} \right)$$

where
- $P^{Spec}(a^{k^{\checkmark}} | s) := \sum_{s' \in S \setminus \{Dead\}} Pr^{Spec}_{s\ s'}(a)$, and $P^{Spec}(a^{k^{\times}} | s) := 1 - P^{Spec}(a^{k^{\checkmark}} | s)$, on a state s,
- $\Delta^{a^{k^{\checkmark}}}_{\tau} := P^C(a^{k^{\checkmark}} | \tau) - P^I(a^{k^{\checkmark}} | \tau)$ and $\Delta^{a^{k^{\times}}}_{\tau} := -\Delta^{a^{k^{\checkmark}}}_{\tau}$.

The proof that the formal definition of $\mathcal{M}_K$ corresponds to the intuition given before Definition 4 is omitted as well of the proof of the following theorem[5].

Theorem 3. *Let $\mathcal{M}_K$ be the MDP induced by "Spec", "Impl", and "Clone". If $\gamma < 1$ or $|\mathcal{M}_K| < \infty$ then $V^\star(\mathsf{i}) \geq 0$ for any policy π, and $V^\star(\mathsf{i}) = 0$ if and only if "Spec" and "Impl" are K-moment equivalent.*

We can now give the definition of the central notion of this paper.

Definition 5. Let "Spec" and "Impl" be two LMPs and $\mathcal{M}_K$ their induced MDP. We define their *K-moment equivalence divergence* as

$$\mathrm{div}_{K\text{-moment}}(\text{"Spec"} \| \text{"Impl"}) := V^\star(\mathsf{i}).$$

3.3 The Choice of the Learning Algorithm and PAC Guaranties

As mentioned in Section 3.1, the full model of the MDP might not be available. Therefore, it is not appropriate to use a Dynamic Programming algorithm such as value iteration [15] to solve the MDP. Instead, we use a Q-Learning algorithm [16]. Q-Learning is an off-policy Temporal Difference (TD) control algorithm which directly approximates $V^\star(\mathsf{i})$. The algorithm has been proven to converge

[4] If one only wants to run a Q-learning algorithm on it, only the model of "Spec" and a possibility of interacting with "Clone" and "Impl" is required.

[5] See `http://www.ift.ulaval.ca/~laviolette/Publications/publications.html`.

to the optimal value [17]. Moreover, some results about its convergence rates have been proposed [6]. However, in the field of verification, the main goals are $(*)$ to find the difference between the implementation of a system and its specification and also $(**)$ to have a guarantee on the fact that this difference is very small in the case where we do not find any such difference during the investigation. Hence, from that perspective, a PAC-guarantee is the most appropriate tool.

Definition 6. We say that we have a *PAC (Probably Approximately Correct) guarantee* for a learning algorithm on an MDP $\mathcal{M}$ if, given an a priori precision $\epsilon > 0$ and a maximal probability error δ, there exists a function $f(\mathcal{M}_K, \epsilon, \delta)$ such that if the number of episodes is greater than $f(\mathcal{M}_K, \epsilon, \delta)$, then

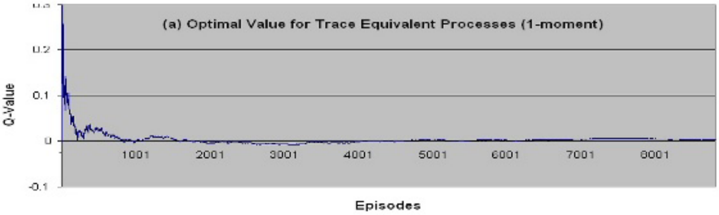
$$Prob\{|\overline{V^{\hat{\pi}}(\mathsf{i})} - V^{\star}(\mathsf{i})| \leq \epsilon\} \geq 1 - \delta \tag{1}$$

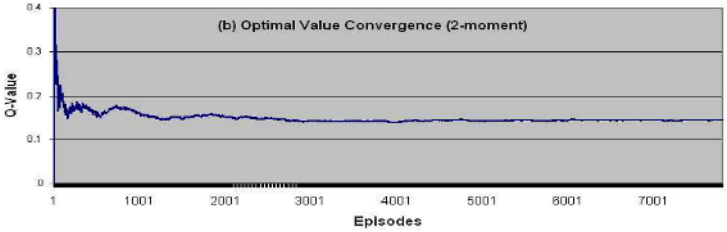
where $\hat{\pi}$ is the policy returned by the Q-learning algorithm and $\overline{V^{\hat{\pi}}(\mathsf{i})}$ is the estimation of $V^{\star}(\mathsf{i})$ given by this algorithm.

The Q-learning algorithm does have a PAC guarantee [12], but the function $f(\mathcal{M}_K, \epsilon, \delta)$ is very difficult to compute, which makes this guarantee unusable in practice. The Fiechter RL algorithm [7] comes with a simpler PAC guarantee and hence one can use it in the current setting. The main drawback of the Fiechter algorithm remains its inefficiency compared to Q-Learning.

However we can still reach goal $(*)$ using any RL learning algorithm. Indeed, when the processes are not K-moment equivalent, we can guarantee a bottom bound for the optimal value using Hoeffding inequality based on the following idea. Let $\hat{\pi}$ be the policy returned by the RL algorithm. Let $\overline{V^{\hat{\pi}}(\mathsf{i})}$ be the estimation of $V^{\hat{\pi}}(\mathsf{i})$ using a Monte Carlo [15] algorithm with m episodes. Given $\epsilon, \delta \in]0,1[$, according to the Hoeffding inequality, if $m \geq \frac{1}{\epsilon^2}\ln(\frac{2}{\delta})$, we have Equation (1) with $V^{\star}(\mathsf{i})$ replaced with $V^{\hat{\pi}}(\mathsf{i})$. Since $V^{\hat{\pi}}(\mathsf{i})$ never exceeds the optimal value $V^{\star}(\mathsf{i})$, we have the PAC guarantee: $Prob\{\overline{V^{\hat{\pi}}(\mathsf{i})} - V^{\star}(\mathsf{i}) \leq \epsilon\} \geq 1 - \delta$.

Experimental Results. The approach described so far has been implemented for the trace and K-moment family equivalences. Two action selection algorithms have been experimented: ϵ-greedy and SoftMax. For both methods, we tried several functions to decrease the ϵ (resp. the τ) values. The combination that produced the best results is SoftMax such that the temperature τ is decreasing from 0.8 to 0.01 according to the function : $\tau = \frac{k}{\text{currentEpisod}+l}$ (k and l are constants). The learning rate α (also called step size) must decrease in order to assure convergence of the Q-Learning algorithm. We tried several decreasing functions and the best convergence results are with $\frac{1}{x}$ where x is the number of times the state-action has been visited. The two following graphics





show how the Q-Learning algorithm converges to the optimal value. In the above graphics, we tracked the optimal value in one execution of 10000 episodes on small examples. Running the algorithm with 1-moment option on trace equivalent processes produces the graphic (a). It is easy to see that the estimated divergence value converges to zero as expected. The second graph (b) is obtained by running the algorithm with 2-moment option and in this case, however, the estimated value converges to a value bigger than zero indicating a difference between the two processes.

4 Other Testable Equivalences

Any equivalence notion that coincides with a recursive replication-free test grammar is compatible with the RL algorithm described earlier. Several known equivalence notions fall in this category. Due to space limitations, we only present the test grammar that we propose for each of these notions, namely, Ready, Failure [10], Barb Acceptance, and Barb Failure [14] equivalences.

Equivalence	Test Grammar
Trace	$\mathcal{T}_{1moment} ::= \omega \mid a.t$
Ready	$\mathcal{T}_{ready} ::= \omega \mid a.t \mid \{a_1, \ldots, a_n\}$
Failure	$\mathcal{T}_{failure} ::= \omega \mid a.t \mid \{\neg a_1, \ldots, \neg a_n\}$
Barb Acceptance	$\mathcal{T}_{BarbAcc} ::= \omega \mid a.t \mid \{a_1, \ldots, a_n\}a.t$
Barb Failure	$\mathcal{T}_{BarbRef} ::= \omega \mid a.t \mid \{\neg a_1, \ldots, \neg a_n\}a.t$

A test of the form $\{a_1, \ldots, a_n\}$ consists in executing the actions $a_1, \ldots, a_n$ on n copies of the current process, whereas the test of the form $\{a_1, \ldots, a_n\}a.t$ consists in executing actions $a_1, \ldots, a_n$ on respective copies of the current process, then executing action a on another copy and if the latter succeeds proceeding with t.

5 Conclusion

The main contributions of this paper are (1) a completely new approach to estimate how far apart two LMPs are and (2) a new family of equivalences (K-moment) that are a good compromise between trace-equivalence and bisimulation. Indeed, we introduce a notion of divergence $\mathrm{div}_{K\text{-moment}}(\,.\,\|\,.\,)$ that can be estimated via some Monte-Carlo estimation using Reinforcement Learning algorithms. Traditional approaches, on the other hand, are based on costly complete calculations on the models. The RL approach therefore opens a way for analyzing huge systems and even infinite ones. Moreover, it can be adapted to other equivalences that can be tested via recursive replication-free test grammars.

For future work, we want to modify the construction of $\mathcal{M}_K$, in order to speed up the calculation, especially for situations like Ready equivalence, where the action set of the MDP is exponentially larger that the one of the LMP. Finally, since the LMP formalism is mathematically quite similar to the MDP, POMDP, and HMM formalisms, the next step will be to apply our approach on these formalisms. In particular, we could contribute to a theory of approximations.

References

1. R. Blute, J. Desharnais, A. Edalat, and P. Panangaden. Bisimulation for labelled Markov processes. In *Proc. of the Twelfth IEEE Symposium On Logic In Computer Science, Warsaw, Poland*, 1997.
2. F. Breugel, S. Shalit, and J. Worrell. Testing labelled Markov processes. In *ICALP*, volume 2380 of *Lecture Notes in Computer Science*, pages 537–548. Springer, 2002.
3. F. Van Breugel and J. Worrell. Approximating and computing behavioural distances in probabilistic transition systems. *Theoretical Computer Science*, 2006.
4. T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley, 1991.
5. J. Desharnais, F. Laviolette, K. Darsini Moturu, and S. Zhioua. Trace equivalence characterization through reinforcement learning. 2006. Accepted for publication in the 19th Canadian Conference on Artificial Intelligence.
6. E. Even-Dar and Y. Mansour. Learning rates for Q-learning. In *COLT '01/EuroCOLT '01: Proc. of the 14th Annual Conference on Computational Learning Theory*, pages 589–604, London, UK, 2001. Springer-Verlag.
7. C. N. Fiechter. *Design and Analysis of Efficient Reinforcement Learning Algorithms.* PhD thesis, Univ. of Pittsburgh, 1997.
8. A. Giacalone, C. Jou, and S. Smolka. Algebraic reasoning for probabilistic concurrent systems. In *Proceedings of the Working Conference on Programming Concepts and Methods*, IFIP TC2, 1990.
9. R.J. Van Glabbeek. The linear time - branching time spectrum ii. In *CONCUR '93: Proceedings of the 4th International Conference on Concurrency Theory*, pages 66–81, London, UK, 1993. Springer-Verlag.
10. C.-C. Jou and S. A. Smolka. Equivalences, congruences, and complete axiomatizations for probabilistic processes. In J. Baeten and J. Klop, editors, *CONCUR 90 1st Int. Conf. on Concurrency Theory*, number 458 in LNCS. Springer-Verlag, 1990.
11. L. P. Kaelbling, M. L. Littman, and A. P. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.
12. M. Kearns and S. Singh. Finite-sample convergence rates for q-learning and indirect algorithms. In *Proc. of the 1998 conference on Advances in neural information processing systems II*, pages 996–1002, Cambridge, MA, USA, 1999. MIT Press.
13. K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.
14. G. Lowe. Representing Nondeterministic and Probabilistic Behaviour in Reactive Processes. Technical report, Progr. Res. Group, Oxford University, 1993.
15. R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1998.
16. C. Watkins.*Learning from Delayed Rewards.* PhD thesis, Univ.of Cambridge, 1989.
17. C. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992.

On Decidability of LTL Model Checking for Process Rewrite Systems

Laura Bozzelli[1], Mojmír Křetínský[2], Vojtěch Řehák[2], and Jan Strejček[2]

[1] Dipartimento di Matematica e Apllicazioni, Università degli Studi di Napoli "Federico II",
Via Cintia, 80126 Napoli, Italy
laura.bozzelli@dma.unina.it

[2] Faculty of Informatics, Masaryk University, Botanická 68a, 60200 Brno, Czech Republic
{kretinsky, rehak, strecek}@fi.muni.cz

Abstract. We establish a decidability boundary of the model checking problem for infinite-state systems defined by *Process Rewrite Systems* (PRS) or *weakly extended Process Rewrite Systems* (wPRS), and properties described by basic fragments of action-based *Linear Temporal Logic* (LTL). It is known that the problem for general LTL properties is decidable for Petri nets and for pushdown processes, while it is undecidable for PA processes. As our main result, we show that the problem is decidable for wPRS if we consider properties defined by formulae with only modalities *strict eventually* and *strict always*. Moreover, we show that the problem remains undecidable for PA processes even with respect to the LTL fragment with the only modality *until* or the fragment with modalities *next* and *infinitely often*.

1 Introduction

Automatic verification of current software systems often needs to model them as infinite-state systems. One of the most powerful formalisms for description of infinite-state systems (except formalisms with Turing power for which nearly all interesting verification problems are undecidable) is called *Process Rewrite Systems* (PRS) [May00]. The PRS framework, based on term rewriting, subsumes many formalisms studied in the context of formal verification, e.g. *Petri nets* (PN), *pushdown processes* (PDA), and process algebras like BPA, BPP, or PA. PRS can be adopted as a formal model for programs with recursive procedures and restricted forms of dynamic creation and synchronization of concurrent processes. A substantial advantage of PRS is that some important verification problems are decidable for the whole PRS class. In particular, Mayr [May00] proved that the *reachability problem* (whether a given state is reachable) and the *reachable property problem* (whether there is a reachable state where some given actions are enabled and some given actions are disabled) are decidable for PRS.

In [KŘS04b], we have presented *weakly extended PRS* (wPRS), where a finite-state control unit with self-loops as the only loops is added to the standard PRS formalism (addition of a general finite-state control unit makes PRS Turing powerful). This control unit enriches PRS by abilities to model a bounded number of arbitrary communication events and global variables whose values are changed only a bounded number of times

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 248–259, 2006.

during any computation. We have proved that the reachability problem remains decidable for wPRS [KŘS04a] and that the problem called *reachability Hennessy–Milner property* (whether there is a reachable state satisfying a given Hennessy–Milner formula) is decidable for wPRS as well [KŘS05]. The hierarchy of all PRS and wPRS classes is depicted in Figure 1.

Concerning the model checking problem, a broad overview of (un)decidability results for subclasses of PRS and various temporal logics can be found in [May98]. Here we focus exclusively on (future) *Linear Temporal Logic* (LTL). It is known that LTL model checking of PDA is EXPTIME-complete [BEM97]. LTL model checking of PN is also decidable, but at least as hard as the reachability problem for PN [Esp94] (the reachability problem is EXPSPACE-hard [May84, Lip76] and no primitive recursive upper bound is known). If we consider only infinite runs, then the problem for PN is EXPSPACE-complete [Hab97, May98].

Conversely, LTL model checking is undecidable for all classes subsuming PA [BH96, May98]. So far, there are only two positive results for these classes. Bouajjani and Habermehl [BH96] have identified a fragment called *simple PLTL*$_\Box$ for which model checking of infinite runs is decidable for PA (strictly speaking, simple PLTL$_\Box$ is not a fragment of LTL as it can express also some non-regular properties, while LTL cannot). Only recently, we have demonstrated that model checking of infinite runs is decidable for PRS and the fragment of LTL capturing exactly fairness properties [Boz05].

Our contribution: This paper completely locates the decidability boundary of the model checking problem for all subclasses of PRS (and wPRS) and all *basic LTL fragments*, where a basic LTL fragment is a set of all formulae containing only a given subset of standard modalities. The boundary is depicted in Figure 2. To locate the boundary, we show the following results.

1. We introduce a new LTL fragment $\mathcal{A}$ and prove that every formula of the basic fragment LTL($\mathsf{F_s}, \mathsf{G_s}$) (i.e. the fragment with modalities *strict eventually* and *strict always* only) can be effectively translated into $\mathcal{A}$. As LTL($\mathsf{F_s}, \mathsf{G_s}$) is closed under negation, we can also translate LTL($\mathsf{F_s}, \mathsf{G_s}$) formulae into negated formulae of $\mathcal{A}$.
2. We show that model checking (of both finite and infinite runs) of wPRS against negated formulae of $\mathcal{A}$ is decidable. The proof employs our results presented in [Boz05, KŘS04a, KŘS05] to reduce the problem to LTL model checking for PDA and PN. Thus we get decidability of model checking for wPRS against LTL($\mathsf{F_s}, \mathsf{G_s}$). Note that LTL($\mathsf{F_s}, \mathsf{G_s}$) is strictly more expressive than the *Lamport logic* (i.e. the basic fragment with modalities *eventually* and *always*), which is again strictly more expressive than the mentioned fragment of fairness properties and also than the *regular* part of simple PLTL$_\Box$.
3. We demonstrate that the model checking problem remains undecidable for PA even if we consider the basic fragment with modality *until* or the basic fragment with modalities *next* and *infinitely often* (which is strictly less expressive than the one with *next* and *eventually*).

The paper is organized as follows. The following section recalls basic definitions. Sections 3, 4, and 5 correspond, respectively, to the three results listed above. The last section discuss other potential applications of our results and it contains an open question driving our future research. Proofs are only sketched due to space constraints. Full proofs can be found in [BKŘS06].

2 Preliminaries

2.1 PRS and Its Extensions

Let $Const = \{X, \dots\}$ be a set of *process constants*. The set of *process terms t* is defined by the abstract syntax $t ::= \varepsilon \mid X \mid t.t \mid t\|t$, where ε is the *empty term*, $X \in Const$, and '.' and '$\|$' mean *sequential* and *parallel compositions*, respectively. We always work with equivalence classes of terms modulo commutativity and associativity of '$\|$', associativity of '.', and neutrality of ε, i.e. $\varepsilon.t = t.\varepsilon = t\|\varepsilon = t$. We distinguish four *classes of process terms* as:

1 – terms consisting of a single process constant, in particular, $\varepsilon \notin 1$,
S – *sequential* terms - terms without parallel composition, e.g. $X.Y.Z$,
P – *parallel* terms - terms without sequential composition, e.g. $X\|Y\|Z$,
G – *general* terms - terms without any restrictions, e.g. $(X.(Y\|Z))\|W$.

Let $M = \{o, p, q, \dots\}$ be a set of *control states*, $\leq$ be a partial ordering on this set, and $Act = \{a, b, c, \dots\}$ be a set of *actions*. Let $\alpha, \beta \in \{1, S, P, G\}$ be classes of process terms such that $\alpha \subseteq \beta$. An (α,β)*-wPRS* (*weakly extended process rewrite system*) Δ is a tuple (R, p_0, X_0), where

- R is a finite set of *rewrite rules* of the form $(p,t_1) \overset{a}{\hookrightarrow} (q,t_2)$, where $t_1 \in \alpha$, $t_1 \neq \varepsilon$, $t_2 \in \beta$, $a \in Act$, and $p, q \in M$ are control states satisfying $p \leq q$,
- the pair $(p_0, X_0) \in M \times Const$ forms the distinguished *initial state*.

By $Act(\Delta)$, $Const(\Delta)$, and $M(\Delta)$ we denote the sets of actions, process constants, and control states occurring in the rewrite rules or the initial state of Δ, respectively.

An (α,β)-wPRS $\Delta = (R, p_0, X_0)$ induces a labelled transition system, whose states are pairs (p,t) such that $p \in M(\Delta)$ is a control state and $t \in \beta$ is a process term over $Const(\Delta)$. The transition relation $\longrightarrow_\Delta$ is the least relation satisfying the following inference rules:

$$\frac{((p,t_1) \overset{a}{\hookrightarrow} (q,t_2)) \in \Delta}{(p,t_1) \xrightarrow{a}_\Delta (q,t_2)} \qquad \frac{(p,t_1) \xrightarrow{a}_\Delta (q,t_2)}{(p,t_1\|t_1') \xrightarrow{a}_\Delta (q,t_2\|t_1')} \qquad \frac{(p,t_1) \xrightarrow{a}_\Delta (q,t_2)}{(p,t_1.t_1') \xrightarrow{a}_\Delta (q,t_2.t_1')}$$

Sometimes we write $\longrightarrow$ instead of $\longrightarrow_\Delta$ if Δ is clear from the context. The transition relation can be extended to finite words over Act in a standard way. To shorten our notation we write pt in lieu of (p,t). A state pt is called *terminal*, written $pt \not\longrightarrow_\Delta$, if there is no state $p't'$ and action a such that $pt \xrightarrow{a}_\Delta p't'$. In this paper we always consider only systems where the initial state is not terminal. A (finite or infinite) sequence

$$\sigma = p_1t_1 \xrightarrow{a_1}_\Delta p_2t_2 \xrightarrow{a_2}_\Delta \dots \xrightarrow{a_n}_\Delta p_{n+1}t_{n+1} \left(\xrightarrow{a_{n+1}}_\Delta \dots \right)$$

is called *derivation over the word* $u = a_1a_2 \dots a_n(a_{n+1} \dots)$ *in* Δ. Finite derivations are also denoted as $p_1t_1 \xrightarrow{u}_\Delta p_{n+1}t_{n+1}$, infinite as $p_1t_1 \xrightarrow{u}_\Delta$. A derivation in Δ is called

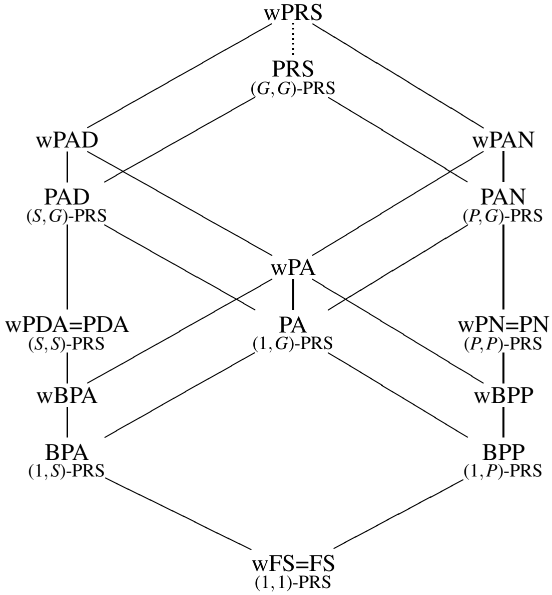


Fig. 1. The hierarchy of PRS and wPRS subclasses

a *run of* Δ if it starts in the initial state p_0X_0 and it is either infinite, or its last state is terminal. Further, $L(\Delta)$ denotes the set of words u such that there is a run of Δ over u.

An (α,β)-wPRS Δ where $M(\Delta)$ is a singleton is called (α,β)-*PRS* (*process rewrite system*) [May00]. In such systems we omit the single control state from rules and states.

Some classes of (α,β)-PRS correspond to widely known models, namely *finite-state systems* (FS), *basic process algebras* (BPA), *basic parallel processes* (BPP), *process algebras* (PA), *pushdown processes* (PDA), and *Petri nets* (PN). The other classes have been named as PAD, PAN, and PRS. The relations between (α,β)-PRS and the mentioned formalisms and names are indicated in Figure 1. Instead of (α,β)-wPRS we juxtapose the prefix 'w-' with the acronym corresponding to the (α,β)-PRS class. For example, we use wBPA rather than $(1,S)$-wPRS. Figure 1 shows the expressiveness hierarchy of all considered classes, where expressive power of a class is measured by the set of transition systems that are definable (up to the strong bisimulation equivalence [Mil89]) by the class. This hierarchy is strict, with a potential exception concerning the classes wPRS and PRS, where the strictness is just our conjecture. For details see [KŘS04b, KŘS04a].

For technical reasons, we define a normal form of wPRS systems. A rewrite rule is *parallel* or *sequential* if it has one of the following forms:

$$\textbf{Parallel rules}:\quad pX_1\|X_2\|\ldots\|X_n \overset{a}{\hookrightarrow} qY_1\|Y_2\|\ldots\|Y_m$$
$$\textbf{Sequential rules}:\quad pX \overset{a}{\hookrightarrow} qY.Z \qquad pX.Y \overset{a}{\hookrightarrow} qZ \qquad pX \overset{a}{\hookrightarrow} qY \qquad pX \overset{a}{\hookrightarrow} q\varepsilon$$

where $X,Y,X_i,Y_j,Z \in Const$, $p,q \in M$, $n>0$, $m \geq 0$, and $a \in Act$. A rule is called *trivial* if it is both parallel and sequential (i.e. it has the form $pX \overset{a}{\hookrightarrow} qY$ or $pX \overset{a}{\hookrightarrow} q\varepsilon$). A wPRS Δ is in *normal form* if it has only parallel and sequential rewrite rules.

2.2 Linear Temporal Logic (LTL) and Studied Problems

The syntax of *Linear Temporal Logic* (LTL) [Pnu77] is defined as follows

$$\varphi ::= tt \mid a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi,$$

where a ranges over Act, X is called *next*, and U is called *until*. The logic is interpreted over infinite as well as nonempty finite words of actions. Given a word $u = u(0)u(1)u(2)\ldots \in Act^* \cup Act^\omega$, $|u|$ denotes the length of the word (we set $|u| = \infty$ if u is infinite). For all $0 \le i < |u|$, by u_i we denote the i^{th} suffix of u, i.e. $u_i = u(i)u(i+1)\ldots$.

The semantics of LTL formulae is defined inductively as follows:

$$\begin{array}{lll}
u \models tt & & \\
u \models a & \text{iff} & u(0) = a \\
u \models \neg\varphi & \text{iff} & u \not\models \varphi \\
u \models \varphi_1 \wedge \varphi_2 & \text{iff} & u \models \varphi_1 \text{ and } u \models \varphi_2 \\
u \models \mathsf{X}\varphi & \text{iff} & |u| > 1 \text{ and } u_1 \models \varphi \\
u \models \varphi_1 \mathsf{U} \varphi_2 & \text{iff} & \exists 0 \le i < |u| \,.\, (u_i \models \varphi_2 \text{ and } \forall 0 \le j < i.\ u_j \models \varphi_1)
\end{array}$$

We say that a nonempty word u *satisfies* φ whenever $u \models \varphi$. Given a set of words L, we write $L \models \varphi$ if $u \models \varphi$ holds for all $u \in L$. We say that a derivation (or run) σ over a word u satisfies φ, written $\sigma \models \varphi$, whenever $u \models \varphi$.

Moreover, we define the following modalities: $\mathsf{F}\varphi$ (*eventually*) standing for $tt\,\mathsf{U}\,\varphi$, $\mathsf{G}\varphi$ (*always*) standing for $\neg\mathsf{F}\neg\varphi$, $\mathsf{F_s}\varphi$ (*strict eventually*) standing for $\mathsf{XF}\varphi$, $\mathsf{G_s}\varphi$ (*strict always*) standing for $\neg\mathsf{F_s}\neg\varphi$, $\overset{\infty}{\mathsf{F}}\varphi$ (*infinitely often*) standing for $\mathsf{GF}\varphi$, $\overset{\infty}{\mathsf{G}}\varphi$ (*almost always*) standing for $\neg\overset{\infty}{\mathsf{F}}\neg\varphi$. Note that $\mathsf{F}\varphi$ is equivalent to $\varphi \vee \mathsf{F_s}\varphi$ but $\mathsf{F_s}\varphi$ cannot be expressed with F as the only modality. Thus $\mathsf{F_s}$ is "stronger" than F. The relation between $\mathsf{G_s}$ and G is similar.

For a set $\{O_1, \ldots, O_n\}$ of modalities, $\mathrm{LTL}(O_1, \ldots, O_n)$ denotes the LTL fragment containing all formulae with modalities $O_1, \ldots, O_n$ only. Such a fragment is called *basic*. Figure 2 shows an expressiveness hierarchy of all studied basic LTL fragments. Indeed, every basic LTL fragment using standard[1] future modalities is equivalent to one of the fragments in the hierarchy, where equivalence between fragments means that every formula of one fragment can be effectively translated into a semantically equivalent formula of the other fragment and vice versa. For example, $\mathrm{LTL}(\mathsf{F_s}, \mathsf{G_s}) \equiv \mathrm{LTL}(\mathsf{F_s})$. Further, the hierarchy is strict. For detailed information about expressiveness of future LTL modalities and LTL fragments we refer to [Str04].

Let $\mathcal{F}$ be an LTL fragment and $\mathcal{C}$ be a class of wPRS systems. The *model checking problem* for $\mathcal{F}$ and $\mathcal{C}$ is to decide whether a given formula $\varphi \in \mathcal{F}$ and a given system $\Delta \in \mathcal{C}$ satisfies $L(\Delta) \models \varphi$. We also mention the problem called *model checking of infinite runs*, where $L(\Delta) \cap Act^\omega \models \varphi$ is examined.

[1] By standard modalities we mean the ones defined in this paper and also other commonly used modalities like *strict until*, *release*, *weak until*, etc. However, it is well possible that one can define a new modality such that there is a basic fragment not equivalent to any of the fragments in the hierarchy.

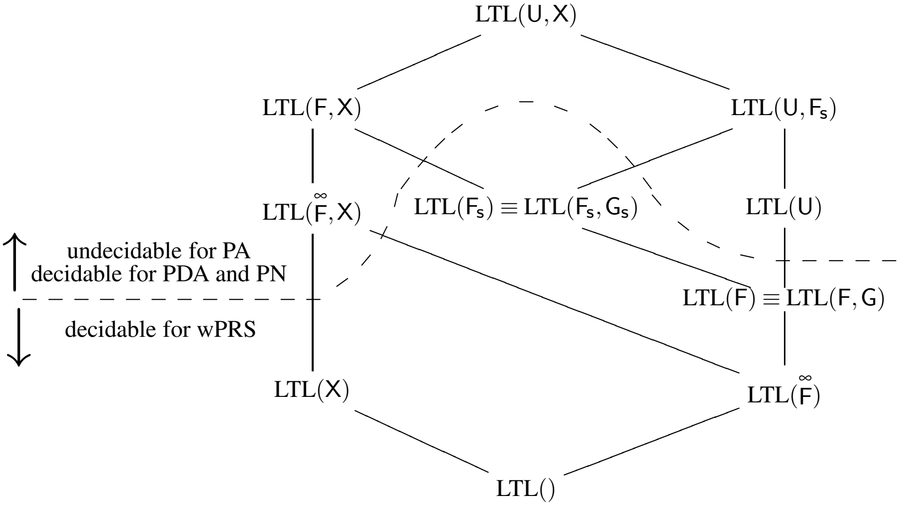


Fig. 2. The hierarchy of basic fragments with model checking decidability boundary

3 Fragment $\mathcal{A}$ and Translation of LTL$(\mathsf{F_s},\mathsf{G_s})$ into $\mathcal{A}$

The $\mathcal{A}$ fragment consists of finite disjunctions of α-*formulae* defined as follows.

Recall that LTL() denotes the fragment of formulae without any modality, i.e. boolean combinations of actions. In the following we use $\varphi_1 \, \mathsf{U}_+ \, \varphi_2$ to abbreviate $\varphi_1 \wedge \mathsf{X}(\varphi_1 \, \mathsf{U} \, \varphi_2)$. Let $\delta = \theta_1 O_1 \theta_2 O_2 \ldots \theta_n O_n \theta_{n+1}$, where $n > 0$, each $\theta_i \in$ LTL(), O_n is '$\wedge \mathsf{G_s}$', and, for each $i < n$, O_i is either 'U' or 'U_+' or '$\wedge \mathsf{X}$'. Further, let $\mathcal{B} \subseteq$ LTL() be a finite set. An α-formula is defined as

$$\alpha(\delta,\mathcal{B}) = \big(\theta_1 O_1 (\theta_2 O_2 \ldots (\theta_n O_n \theta_{n+1}) \ldots)\big) \wedge \bigwedge_{\psi \in \mathcal{B}} \mathsf{G_s F_s} \psi$$

Hence, a word u satisfies $\alpha(\delta,\mathcal{B})$ iff u can be written as $u_1.u_2.\cdots.u_{n+1}$, where

- each u_i consists only of actions satisfying θ_i and
  - $|u_i| \geq 0$ if $i = n+1$ or O_i is 'U',
  - $|u_i| > 0$ if O_i is 'U_+',
  - $|u_i| = 1$ if O_i is '$\wedge \mathsf{X}$' or '$\wedge \mathsf{G_s}$',
- and u_{n+1} satisfies $\mathsf{G_s F_s} \psi$ for every $\psi \in \mathcal{B}$.

Proof of the following lemma is a simple exercise.

Lemma 1. *A conjunction of* α*-formulae can be effectively converted into an equivalent disjunction of* α*-formulae.*

Theorem 2. *Every LTL*$(\mathsf{F_s},\mathsf{G_s})$ *formula can be translated into an equivalent disjunction of* α*-formulae.*

Proof (Sketch). Given an LTL($\mathsf{F_s}, \mathsf{G_s}$) formula φ, we construct a finite set A_φ of α-formulae such that φ is equivalent to disjunction of formulae in A_φ. The proof proceeds by induction on the length of φ. The base case shows that the theorem holds for all formulae of LTL(). The inductive step is done by a detailed analysis of the structure of φ (it distinguishes 19 cases). □

4 Model Checking of wPRS Against Negated $\mathcal{A}$

This section is devoted to decidability of the model checking problem for wPRS and negated formulae of the $\mathcal{A}$ fragment. In fact, we prove decidability of the dual problem, i.e. whether a given wPRS system has a run satisfying a given formula of $\mathcal{A}$. Finite and infinite runs are treated separately.

Theorem 3. *The problem whether a given wPRS system has a finite run satisfying a given* α*-formula is decidable.*

Proof (Sketch). The problem is reduced to the reachability Hennessy–Milner property problem, which is decidable for wPRS [KŘS05]. □

The problem for infinite runs is more complicated. In order to solve it, we introduce more terminology and notation. First we define β-*formulae* and regular languages called γ-*languages*. Let $w = a_1 O_1 a_2 O_2 \ldots a_n O_n$, where $n \geq 0$, $a_1, \ldots, a_n \in Act$ are pairwise distinct actions and each O_i is either 'U_+' or '$\wedge\mathsf{X}$'. Further, let $B \subseteq Act \smallsetminus \{a_1, \ldots, a_n\}$ be a nonempty finite set of actions and $C \subseteq B$. A β-formula $\beta(w,B,C)$ and γ-language $\gamma(w,C)$ are defined as

$$\beta(w,B,C) = \big(a_1 O_1 (a_2 O_2 \ldots (a_n O_n \mathsf{G} \bigvee_{b \in B} b) \ldots)\big) \wedge \bigwedge_{b \in C} \mathsf{GF}b \wedge \bigwedge_{b \in B \smallsetminus C} (\mathsf{F}b \wedge \neg\mathsf{GF}b)$$

$$\gamma(w,C) = a_1^{o_1}.a_2^{o_2}.\cdots.a_n^{o_n}.L,$$

$$\text{where } o_i = \begin{cases} + & \text{if } O_i = \mathsf{U}_+ \\ 1 & \text{if } O_i = \wedge\mathsf{X} \end{cases} \text{ and } L = \begin{cases} \{\varepsilon\} & \text{if } C = \emptyset \\ \bigcap_{b \in C} C^*.b.C^* & \text{otherwise} \end{cases}$$

Roughly speaking, a β-formula is a more restrictive version of an α-formula and in context of β-formulae we consider infinite words only. Contrary to δ of an α-formula, w of a β-formula employs actions rather than LTL() formulae. While a tail of an infinite word satisfying an α-formula is specified by θ_{n+1}, in the definition of β-formulae we use a set B containing exactly all the actions of the tail and its subset C of exactly all actions occurring infinitely many times in the tail.

Note that an infinite word satisfies a formula $\beta(w,B,C)$ if and only if it can be divided into a prefix $u \in \gamma(w,B)$ and a suffix $v \in C^\omega$ such that v contains infinitely many occurrences of every $c \in C$.

Let w, B, C be defined as above. We say that a finite derivation σ over a word u *satisfies* $\gamma(w,C)$ if and only if $u \in \gamma(w,C)$. We write $(w',B') \sqsubseteq (w,B)$ whenever $B' \subseteq B$ and $w' = a_{i_1} O_{i_1} a_{i_2} O_{i_2} \ldots a_{i_k} O_{i_k}$ for some $1 \leq i_1 < i_2 < \ldots < i_k \leq n$. Moreover, we write $(w',B',C') \sqsubseteq (w,B,C)$ whenever $(w',B') \sqsubseteq (w,B)$, B' is nonempty, and $C' \subseteq C \cap B'$.

Remark 4. If u is an infinite word satisfying $\beta(w,B,C)$ and v is an infinite *subword* of u (i.e. it arises from u by omitting some letters), then there is exactly one triple $(w',B',C') \sqsubseteq (w,B,C)$ such that $v \models \beta(w',B',C')$. Further, for each finite subword v of u, there is exactly one pair (w',B') such that $(w',B') \sqsubseteq (w,B)$ and $v \in \gamma(w',B')$.

Given a PRS in normal form, by $tri(\Delta)$, $par(\Delta)$, and $seq(\Delta)$ we denote the system Δ restricted to trivial, parallel, and sequential rules, respectively. A derivation in $tri(\Delta)$ is called a *trivial* derivation in Δ. In the following we write simply tri, par, seq as Δ is always clearly determined by the context.

Definition 5. *Let Δ be a PRS in normal form and $\beta(w,B,C)$ be a β-formula. The PRS Δ is in* flat (w,B,C)-form *if and only if for each $X,Y \in Const(\Delta)$, each $(w',B',C') \sqsubseteq (w,B,C)$, and each $B'' \subseteq B$, the following conditions hold:*

1. *If there is a finite derivation $X \xrightarrow{u} Y$ satisfying $\gamma(w',B'')$, then there is also a finite derivation $X \xrightarrow{v}_{tri} Y$ satisfying $\gamma(w',B'')$.*
2. *If there is a term t and a finite derivation $X \xrightarrow{u} t$ satisfying $\gamma(w',B'')$, then there is also a constant Z and a finite derivation $X \xrightarrow{v}_{tri} Z$ satisfying $\gamma(w',B'')$.*
3. *If $w' = \varepsilon$ and there is an infinite derivation $X \xrightarrow{u}$ satisfying $\beta(w',B',C')$, then there is also an infinite derivation $X \xrightarrow{v}_{tri}$ satisfying $\beta(w',B',C')$.*
4. *If there is an infinite derivation $X \xrightarrow{u}_{par}$ satisfying $\beta(w',B',C')$, then there is also an infinite derivation $X \xrightarrow{v}_{tri}$ satisfying $\beta(w',B',C')$;*
5. *If there is an infinite derivation $X \xrightarrow{u}_{seq}$ satisfying $\beta(w',B',C')$, then there is also an infinite derivation $X \xrightarrow{v}_{tri}$ satisfying $\beta(w',B',C')$.*

Intuitively, the system is in flat (w,B,C)-form if for every derivation of one of the listed types there is an "equivalent" trivial derivation. All conditions of the definition can be checked due to the following lemma, [Boz05], and decidability of LTL model checking for PDA and PN. Lemma 7 says that every PRS in normal form can be transformed into an "equivalent" flat system. Finally, the Lemma 10 says that if a PRS system in flat (w,B,C)-form has an infinite derivation satisfying $\beta(w,B,C)$, then it has also a trivial infinite derivation satisfying $\beta(w,B,C)$. Note that it is easy to check whether such a trivial derivation exists.

Lemma 6. *Given a γ-language $\gamma(w,C)$, a PRS system Δ, and constants X,Y, the following problems are decidable:*
(i) Is there any derivation $X \xrightarrow{u} Y$ satisfying $\gamma(w,C)$?
(ii) Is there any derivation $X \xrightarrow{u} t$ such that t is a term and $u \in \gamma(w,C)$?

Proof (Sketch). Both problems can be reduced to the reachability problem for wPRS, which is known to be decidable [KŘS04a]. □

The proof of the following lemma contains the algorithmic core of this section.

Lemma 7. *Let Δ be a PRS in normal form and $\beta(w,B,C)$ be a β-formula. One can construct a PRS Δ' in flat (w,B,C)-form such that for each $(w',B',C') \sqsubseteq (w,B,C)$ and each $X \in Const(\Delta)$, Δ' is equivalent to Δ with respect to the existence of an infinite derivation starting from X and satisfying $\beta(w',B',C')$.*

Proof (Sketch). All conditions of Definition 5 can be checked for each $X,Y \in Const(\Delta)$, each $(w',B',C') \sqsubseteq (w,B,C)$, and each $B'' \subseteq B$. For Conditions 1 and 2, this follows from Lemma 6. The problem whether there is an infinite derivation $X \xrightarrow{u}$ satisfying $\beta(\varepsilon,B',C')$ is a special case of the *fairness problem*, which is decidable due to [Boz05]. Finally, Conditions 4 and 5 can be checked due to decidability of LTL model checking for PDA and PN. If there is a non-satisfied condition, we add some trivial rules forming the missing derivation. □

Definition 8 (Subderivation). *Let Δ be a PRS in normal form and σ_1 be a (finite or infinite) derivation $s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \ldots$, where $s_1 \xrightarrow{a_1} s_2$ has the form $X \xrightarrow{a_1} Y.Z$ and, for each $i \geq 2$, if s_i is not the last state of the derivation, then it has the form $s_i = t_i.Z$ with $t_i \neq \varepsilon$. Then σ_1 is called a* subderivation *of a derivation σ if σ has a suffix σ' satisfying the following:*

1. *every transition step in σ' is of the form $s_i\|t' \xrightarrow{a_i} s_{i+1}\|t'$ or $s_i\|t' \xrightarrow{b} s_i\|t''$, where $t' \xrightarrow{b} t''$,*
2. *in σ', if we replace every step of the form $s_i\|t' \xrightarrow{a_i} s_{i+1}\|t'$ by step $s_i \xrightarrow{a_i} s_{i+1}$ and we skip every step of the form $s_i\|t' \xrightarrow{b} s_i\|t''$, we get precisely σ_1.*

Further, if σ_1 and σ are finite, the last term of σ_1 is a process constant, and σ is a prefix of a derivation σ', then σ_1 is also a subderivation *of σ'.*

Remark 9. Let Δ be a PRS in normal form and σ be a derivation of Δ having a suffix σ' of the form $\sigma' = X\|t \xrightarrow{a} (Y.Z)\|t \xrightarrow{u}$. Then, there is a subderivation of σ whose first transition step $X \xrightarrow{a} Y.Z$ corresponds to the first transition step of σ'.

Intuitively, the subderivation captures the behaviour of the subterm $Y.Z$ since its emergence until it is possibly reduced to a term without any sequential composition. Due to the normal form of Δ, the subterm $Y.Z$ behaves undependently on the rest of the term (as long as it contains a sequential composition).

Lemma 10. *Let Δ be a PRS in flat (w,B,C)-form. Then, the following condition holds for each $X \in Const(\Delta)$ and each $(w',B',C') \sqsubseteq (w,B,C)$:*
If there is an infinite derivation $X \xrightarrow{u}$ satisfying $\beta(w',B',C')$, then there is also an infinite derivation $X \xrightarrow{v}_{tri}$ satisfying $\beta(w',B',C')$.

Proof (Sketch). Given an infinite derivation σ satisfying a formula $\beta(\sigma) = \beta(w',B',C')$ where $(w',B',C') \sqsubseteq (w,B,C)$, by *trivial equivalent* of σ we mean an infinite trivial derivation starting in the same term as σ and satisfying $\beta(\sigma)$. Similarly, given a finite derivation σ satisfying some $\gamma(\sigma) = \gamma(w',B')$ where $(w',B') \sqsubseteq (w,B)$, by *trivial equivalent* of σ we mean a finite trivial derivation σ' such that σ' starts in the same term as σ, it satisfies $\gamma(\sigma)$, and if the last term of σ is a process constant, then the last term of σ' is the same process constant.

The lemma is proven by contradiction. We assume that there exist some infinite derivations violating the condition of the lemma. Let σ be one of these derivations such that the number of transition steps of σ generated by sequential non-trivial rules with actions $a \notin B$ is minimal (note that this number is always finite as we consider derivations satisfying $\beta(w',B',C')$ for some $(w',B',C') \sqsubseteq (w,B,C)$). First, we prove that every subderivation of σ has a trivial equivalent. Then we replace all subderivations of σ by the corresponding trivial equivalents. This step is technically nontrivial because σ may have infinitely many subderivations. By the replacement we obtain an infinite derivation σ' satisfying $\beta(\sigma)$ and starting in the same process constant as σ. Moreover, σ' has no subderivations and hence it does not contain any sequential operator. Flat (w,B,C)-form of Δ (Condition 4) implies that σ' has a trivial equivalent. This is also a trivial equivalent of σ which means that σ does not violate the condition of our lemma. □

Theorem 11. *The problem whether a given PRS Δ in normal form has an infinite run satisfying a given formula $\beta(w,B,C)$ is decidable.*

Proof. Due to Lemmata 7 and 10, the problem can be reduced to the problem whether there is an infinite derivation $X \xrightarrow{v}_{tri}$ satisfying $\beta(w,B,C)$. This problem corresponds to LTL model checking of finite-state systems, which is decidable. □

The following three steps show that the previous theorem holds even for wPRS and α-formulae. The corresponding theorems and proofs can be found in [BKŘS06].

1. First we prove that the theorem holds even for α-formulae. In the proof we assign a fresh action a_θ to each subformula $\theta \in \mathrm{LTL}()$ of a given α-formula. For every such θ and every rule $t_1 \overset{a}{\hookrightarrow} t_2$ of a given PRS in normal form, if $a \models \theta$ then we add a rule $t_1 \overset{a_\theta}{\hookrightarrow} t_2$. Now we replace every θ in the α-formula by a corresponding action a_θ. The system with added rules has a run satisfying the resulting formula iff the original system has a run satisfying the original α-formula. Moreover, the resulting formula can be easily transformed into a β-formula.
2. Now we show that the system Δ does not have to be in normal form. The proof uses a modification of the standard algorithm transforming a general PRS system into an 'equivalent' PRS system in normal form [May00].
3. The last step is to move from PRS to wPRS. To remove control states from the wPRS system Δ, we replace every rule $pt_1 \overset{a}{\hookrightarrow} pt_2$ by the rule $pt_1 \overset{a_p}{\hookrightarrow} pt_2$ and every rule $pt_1 \overset{a}{\hookrightarrow} qt_2$ by the rule $pt_1 \overset{a_{p<q}}{\hookrightarrow} qt_2$. In a given α-formula, we replace every action a with $\bigvee_{p,q \in M(\Delta)}(a_p \vee a_{p<q})$. Let Δ' be the resulting PRS system and α' the resulting α-formula. We define a finite set U of α-formulae such that a run of Δ' satisfies some formula of U iff it corresponds to a correct behaviour of control unit. Hence, Δ has a run satisfying the original α-formula iff Δ' has a run satisfying α' in conjunction with one of the α-formulae of U. As conjunction of two α-formulae can be transformed into equivalent disjunction of α-formulae, we are done.

Theorem 12. *The problem whether a given wPRS system has an infinite run satisfying a given α-formula is decidable.*

As $\mathrm{LTL}(\mathsf{F_s},\mathsf{G_s})$ is closed under negation, Theorems 2, 3, and 12 give us the following.

Corollary 13. *The model checking problem for wPRS and LTL$(\mathsf{F_s},\mathsf{G_s})$ is decidable.*

This problem is EXPSPACE-hard due to EXPSPACE-hardness of the model checking problem for $\mathrm{LTL}(\mathsf{F},\mathsf{G})$ for PN [Hab97]. Our decidability proof does not provide any primitive recursive upper bound as it employs LTL model checking for PN, for which no primitive recursive upper bound is known.

5 Undecidability Results

Obviously, the model checking for wPRS and $\mathrm{LTL}(\mathsf{X})$ is decidable. Hence, to prove that the decidability boundary of Figure 2 is drawn correctly, it remains to show the following.

Theorem 14. *Model checking of PA against LTL*(U) *is undecidable. Model checking of PA against LTL*$(\overset{\infty}{\mathsf{F}},\mathsf{X})$ *is undecidable as well.*

Proof (Sketch). In both cases, the proof is done by reduction from the non-halting problem for Minsky 2-counter machine. □

In the proof of the previous theorem, the PA systems constructed there have only infinite runs. This means that model checking of infinite runs remains undecidable for PA and both $\mathrm{LTL}(\overset{\infty}{\mathsf{F}},\mathsf{X})$ and $\mathrm{LTL}(\mathsf{U})$.

6 Conclusion

We have established the decidability border of model checking of wPRS classes and basic fragments of future LTL (see Figure 2) by showing that the model checking problem of wPRS against $\mathrm{LTL}(\mathsf{F_s},\mathsf{G_s})$ is decidable, while the same problem for PA and $\mathrm{LTL}(\mathsf{U})$ or $\mathrm{LTL}(\mathsf{X},\overset{\infty}{\mathsf{F}})$ is undecidable. So far, only two positive results on LTL model checking of PA (and classes subsuming PA) have been published: decidability of model checking of infinite runs for PRS and LTL fragment of fairness properties [Boz05] and decidability of the same problem for PA and *simple* $PLTL_{\Box}$ [BH96]. Note that the fairness fragment and the regular part of simple $\mathrm{PLTL}_{\Box}$ are strictly less expressive than $\mathrm{LTL}(\mathsf{F},\mathsf{G})$ (also known as Lamport logic), which is again strictly less expressive than $\mathrm{LTL}(\mathsf{F_s},\mathsf{G_s})$. We also emphasize that our positive result for $\mathrm{LTL}(\mathsf{F_s},\mathsf{G_s})$ deals with both finite and infinite runs, and with wPRS rather than PRS or PA only.

It is also worth mentioning that our proof techniques differ from those used in [Boz05] and [BH96]. The decidability proof for $\mathrm{LTL}(\mathsf{F_s},\mathsf{G_s})$ is based on the auxiliary result saying that model checking for wPRS and negated $\mathcal{A}$ fragment is decidable. In fact, this auxiliary result is very powerful. We conjecture that it also implies decidability of the model checking problem of wPRS and the common fragment of CTL and LTL called $\mathrm{LTL}^{\mathrm{det}}$ [Mai00]. Note that $\mathrm{LTL}^{\mathrm{det}}$ is semantically incomparable with $\mathrm{LTL}(\mathsf{F_s},\mathsf{G_s})$.

Unfortunately, our results are insufficient to establish the decidability border for basic LTL fragments with both future and past modalities. Indeed, fragments $\mathrm{LTL}(\mathsf{F_s},\mathsf{P_s})$ and $\mathrm{LTL}(\mathsf{F},\mathsf{P})$, where $\mathsf{P},\mathsf{P_s}$ are past counterparts of $\mathsf{F},\mathsf{F_s}$ respectively, do not semantically coincide with any fragment of Figure 2 and decidability of the model checking problem for these two fragments and all wPRS classes subsuming PA is an open question. However, we conjecture that our technique can be adopted to answer this question positively.

Acknowledgment. Authors have been partially supported as follows: M. Křetínský by the Grant Agency of the Czech Republic, grant No. 201/06/1338, V. Řehák by the research centre "Institute for Theoretical Computer Science (ITI)", project No. 1M0545, and J. Strejček by the Academy of Sciences of the Czech Republic, grant No. 1ET408050503. The paper has been written during J. Strejček's postdoc stay in LaBRI, Univeristé Bordeaux 1.

References

[BEM97] A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *Proc. of CONCUR'97*, volume 1243 of *LNCS*, pages 135–150, 1997.

[BH96] A. Bouajjani and P. Habermehl. Constrained properties, semilinear systems, and petri nets. In *Proc. of CONCUR'96*, volume 1119 of *LNCS*, pages 481–497. Springer, 1996.

[BKŘS06] L. Bozzelli, M. Křetínský, V. Řehák, and J. Strejček. On Decidability of LTL Model Checking for Weakly Extended Process Rewrite Systems. Technical Report FIMU-RS-2006-05, Faculty of Informatics, Masaryk University, Brno, 2006. A full version of the paper presented at FSTTCS'06.

[Boz05] L. Bozzelli. Model checking for process rewrite systems and a class of action-based regular properties. In *Proc. of VMCAI'05*, volume 3385 of *LNCS*, pages 282–297. Springer, 2005.

[Esp94] J. Esparza. On the decidability of model checking for several mu-calculi and petri nets. In *CAAP*, volume 787 of *LNCS*, pages 115–129. Springer, 1994.

[Hab97] P. Habermehl. On the complexity of the linear-time μ-calculus for Petri nets. In *Proceedings of ICATPN'97*, volume 1248 of *LNCS*, pages 102–116. Springer, 1997.

[KŘS04a] M. Křetínský, V. Řehák, and J. Strejček. Extended process rewrite systems: Expressiveness and reachability. In *Proceedings of CONCUR'04*, volume 3170 of *LNCS*, pages 355–370. Springer, 2004.

[KŘS04b] M. Křetínský, V. Řehák, and J. Strejček. On extensions of process rewrite systems: Rewrite systems with weak finite-state unit. In *Proceedings of INFINITY'03*, volume 98 of *ENTCS*, pages 75–88. Elsevier, 2004.

[KŘS05] M. Křetínský, V. Řehák, and J. Strejček. Reachability of Hennessy-Milner properties for weakly extended PRS. In *Proceedings of FSTTCS 2005*, volume 3821 of *LNCS*, pages 213–224. Springer, 2005.

[Lip76] R. Lipton. The reachability problem is exponential-space hard. Technical Report 62, Department of Computer Science, Yale University, 1976.

[Mai00] M. Maidl. The common fragment of CTL and LTL. In *Proc. 41th Annual Symposium on Foundations of Computer Science*, pages 643–652, 2000.

[May84] Ernst W. Mayr. An algorithm for the general Petri net reachability problem. *SIAM Journal on Computing*, 13(3):441–460, 1984.

[May98] R. Mayr. *Decidability and Complexity of Model Checking Problems for Infinite-State Systems*. PhD thesis, Technische Universität München, 1998.

[May00] R. Mayr. Process rewrite systems. *Information and Computation*, 156(1):264–286,2000.

[Mil89] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Min67] Marvin L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, 1967.

[Pnu77] A. Pnueli. The temporal logic of programs. In *Proc. 18th IEEE Symposium on the Foundations of Computer Science*, pages 46–57, 1977.

[Str04] J. Strejček. *Linear Temporal Logic: Expressiveness and Model Checking*. PhD thesis, Faculty of Informatics, Masaryk University in Brno, 2004.

Monitoring of Real-Time Properties

Andreas Bauer, Martin Leucker, and Christian Schallhart

Institut für Informatik, Technische Universität München, Germany

Abstract. This paper presents a construction for runtime monitors that check real-time properties expressed in timed LTL (TLTL). Due to D'Souza's results, TLTL can be considered a natural extension of LTL towards real-time. Moreover, a typical obstacle in runtime verification is solved both for untimed and timed formulae, in that standard models of linear temporal logic are infinite traces, whereas in runtime verification only finite system behaviours are at hand. Therefore, a 3-valued semantics (*true, false, inconclusive*) for LTL and TLTL on finite traces is defined that resembles the infinite trace semantics in a suitable and intuitive manner. Then, the paper describes how to construct, given a (T)LTL formula, a deterministic monitor with three output symbols that reads a finite trace and yields its according 3-valued (T)LTL semantics. Notably, the monitor rejects a trace as early as possible, in that any minimal bad prefix results in *false* as a return value.

1 Introduction

Runtime verification [9] is becoming a popular tool to complement verification techniques such as model checking and testing, especially for so-called black box systems. In a nutshell, runtime verification works as follows. A correctness property φ, usually formulated in some linear temporal logic, such as LTL [20], is given and a so-called *monitor* that accepts all models for φ is automatically generated. The system under scrutiny as well as the generated monitor are then executed in parallel, such that the monitor observes the system's behaviour. System behaviour which violates property φ is then detected by the monitor and an according alarm signal is returned.

Monitors can be employed in different phases of system development: In the testing phase [7], the system is executed with typical inputs and monitors are observed for complaints. At customer's site, monitors check for bugs that escaped the testing phase and may trigger recovery actions [5].

Various runtime verification approaches for LTL have been proposed already [13,16,17,15,23]. However, the current approaches suffer—to our opinion—from the treatment of the following obstacle: The semantics of LTL is defined over infinite (behavioural) traces whereas monitoring a running system allows an at most finite view. In consequence, various authors have proposed custom interpretations of LTL over finite traces using *weak* and *strong semantics*: the weak interpretation of a formula φ w. r. t. a finite trace u is that if up to the point where u ends, "nothing has yet gone wrong", φ holds. In the strong view, φ holds only if it evaluates to $true$ within u. (see [12] for an overview). However, good examples can be found for each of the interpretations and—at the same time—also examples that the chosen approach is misleading.

In this paper, we propose a simple, yet—as we find—convincing way to overcome this obstacle. Instead of trying to define a two-valued semantics for LTL on finite traces,

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 260–272, 2006.

we define a three valued semantics, using values $true$, *false*, and ?, where the latter denotes *inconclusive*. Given a finite string u and a formula φ, the truth values are defined as expected: if there is no continuation of u satisfying φ, the value is *false*. If every continuation of u satisfies φ, we go for $true$. Otherwise, we say ?, since the observations so far are just inconclusive to say either $true$ or *false*.

We argue that it is important to work with three instead of two truth values: Consider, for instance, the property $G\neg p$ stating that no state satisfying p should occur. Clearly, when p is observed, the monitor should complain. As long as p does not hold, it is misleading to say that the formula is $true$, since the next observation might already violate the formula. On the other hand, consider the formula $\neg p\, U\, init$ stating that nothing bad (p) should happen before the init function is called. If, indeed, the init function has been called and no p has been observed before, the formula is $true$, regardless what will happen in the future. For testing and verification, it is important to know whether some property is indeed $true$ or whether the current observation is just inconclusive.

Thus, in this paper, we propose a 3-valued logic, LTL$_3$, which can be interpreted over finite traces based on the standard semantics of LTL for infinite traces. Furthermore, we describe how to construct, given an LTL formula, a (deterministic) finite state machine (FSM) with three output symbols. This automaton reads finite traces and yields their 3-valued LTL semantics. Hence, it can be directly deployed for runtime verification. Standard minimisation techniques for FSMs can be used to obtain an *optimal* FSM w. r. t. number of states.

Our 3-valued semantics for LTL rounds off the study of safety properties in terms of automata in [18] from a temporal logic perspective. In [18], a *bad prefix* (of a Büchi automaton), is defined as a finite prefix which cannot be the prefix of any accepting trace. Dually, a *good prefix* is a finite prefix such that any infinite extension of the trace will be accepted. It is exactly this classification that forms the basis of our 3-valued semantics: "bad prefixes" (of formulas) are mapped to *false*, "good prefixes" evaluate to $true$, while the remaining prefixes yield ?. Thus, monitors for 3-valued formulas classify prefixes as one of $good = true$, $bad = false$, or ? (neither *good* nor *bad*).

Since an extension of a bad (good) prefix is bad (good, respectively), there is a *minimal* bad (good) prefix for every bad (good) prefix. In runtime verification, one is interested in getting information already for minimal prefixes and one solution was worked out in [10]. However, all "bad prefixes" for a formula φ gives rise to *false*–also minimal ones. Thus, the correctness of our monitor procedures for LTL and TLTL ensures that already for minimal good or bad prefixes one of $true$ or *false* is obtained. Altogether, we get a coherent study of (not only safety) LTL properties based on finite prefixes together with *optimal* acceptors, as they are called in [10], based on elementary results for LTL and automata theory.

To make our result easily accessible to the reader and to complete the picture started in [10], our concepts are first developed in the setting of LTL. However, the main concern of this paper are real-time systems. Therefore, we develop our ideas also for TLTL, a logic introduced in [21], which, as argued by D'Souza in [11] can be considered a natural counterpart of LTL in the timed setting. Hence, for a TLTL formula a monitor is constructed which operates over finite *timed* traces. Again, by correctness of our construction, monitors signal faults or satisfaction "as early as possible". While the general scheme, as we show, is also applicable in the timed setting, the monitor construction is technically much more involved. Automata for TLTL employ so-called *event recording* and *event predicting* clocks. Since in runtime verification the future of a trace is not known, predicting events are difficult to handle. We introduce *symbolic runs* and show their benefit for checking promises efficiently, avoiding the translation of event-clock automata to (predicting-free) timed automata.

[14] studies monitor generation based on LTL enriched with a freeze quantifier for time. In [24,6], *fault diagnosis* for timed systems is examined, a problem that is more complicated than runtime verification. However, only timed automata or event recording automata are used, no prediction of events is supported. TLTL is event-based, meaning that the system emits events when the system's state has changed. In [19] monitoring of continuous signals is considered, which is intrinsicly different to observing discrete signals in a continuous time domain. All of the work mentioned so far employs a 2-valued semantics. In [10], minimal prefixes for runtime verification are discussed, which our approach offers for free thanks to the 3-valued semantics.

We have implemented the untimed setting and validated our approach examining a real-world case study. The monitor generator, exemplifying material, a case study, and a full version of the paper is available from `http://runtime.in.tum.de/`.

2 Preliminaries

For the remainder of this paper, let AP be a finite set of atomic propositions and $\Sigma = 2^{\mathrm{AP}}$ a finite alphabet. We write a_i for any single element of Σ, i.e., a_i is a possibly empty set of propositions taken from AP. Finite traces over Σ are elements of Σ^*, and are usually denoted by $u, u', u_1, u_2, \ldots$, whereas infinite traces are elements of Σ^ω, usually denoted by $w, w', w_1, w_2, \ldots$.

The set of LTL formulae is inductively defined by the following grammar:

$$\varphi ::= true \mid p \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\, U\, \varphi \mid X\varphi \quad (p \in \mathrm{AP})$$

Let $i \in \mathbb{N}$ be a position. The semantics of LTL formulae is defined inductively over infinite sequences $w = a_0 a_1 \ldots \in \Sigma^\omega$ as follows: $w, i \models true$, $w, i \models \neg\varphi$ iff $w, i \not\models \varphi$, $w, i \models p$ iff $p \in a_i$, $w, i \models \varphi_1 \vee \varphi_2$ iff $w, i \models \varphi_1$ or $w, i \models \varphi_2$, $w, i \models \varphi_1 U \varphi_2$ iff there exists $k \geq i$ with $w, k \models \varphi_2$ and for all l with $i \leq l < k$, $w, l \models \varphi_1$, and $w, i \models X\varphi$ iff $w, i+1 \models \varphi$. Further, let $w \models \varphi$, iff $w, 0 \models \varphi$. For every LTL formula φ, its set of models, denoted by $\mathcal{L}(\varphi)$, is a regular set of infinite traces and can be described by a corresponding Büchi automaton.

A (nondeterministic) Büchi automaton (NBA) is a tuple $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$, where Σ is a finite alphabet, Q is a finite non-empty set of states, $Q_0 \subseteq Q$ is a set of initial states, $\delta : Q \times \Sigma \to 2^Q$ is the transition function, and $F \subseteq Q$ is a set of accepting states. We extend the transition function $\delta : Q \times \Sigma \to 2^Q$, as usual, to $\delta' : 2^Q \times \Sigma^* \to 2^Q$ by $\delta'(Q', \epsilon) = Q'$where $Q' \subseteq Q$and $\delta'(Q', ua) = \bigcup_{q' \in \delta'(Q', u)} \delta(q', a)$. To simplify notation, we use δ for both δ and δ'. A NBA is called *deterministic* iff for all $q \in Q$, $a \in \Sigma$, $|\delta(q, a)| = 1$, and $|Q_0| = 1$. We use DBA to denote a deterministic Büchi automaton. A *run* of an automaton $\mathcal{A}$ on a word $w = a_1 \ldots \in \Sigma^\omega$ is a sequence of states and actions $\rho = q_0 a_1 q_1 \ldots$, where q_0 is an initial state of $\mathcal{A}$ and for all $i \in \mathbb{N}$

we have $q_{i+1} \in \delta(q_i, a_i)$. For a run ρ, let $\text{Inf}(\rho)$ denote the states visited infinitely often. A run ρ of a NBA $\mathcal{A}$ is called *accepting* iff $\text{Inf}(\rho) \cap F \neq \emptyset$.

A nondeterministic *finite automaton* (NFA) $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ is one where Σ, Q, Q_0, δ, and F are defined as for a Büchi automaton, but which operates on finite words. A *run* of $\mathcal{A}$ on a word $w = a_1 \ldots a_n \in \Sigma^*$ is a sequence of states and actions $\rho = q_0 a_1 q_1 \ldots q_n$, where q_0 is an initial state of $\mathcal{A}$ and for all $i \in \mathbb{N}$ we have $q_{i+1} \in \delta(q_i, a_i)$. The run is called accepting if $q_n \in F$. A NFA is called *deterministic* iff for all $q \in Q$, $a \in \Sigma$, $|\delta(q, a)| = 1$, and $|Q_0| = 1$. We use DFA to denote a deterministic finite automaton.

Finally, let us recall the notion of a *Moore machine*, also called *finite-state machine* (FSM), which is a finite state automaton enriched with output, formally denoted by a tuple $(\Sigma, Q, Q_0, \delta, \Delta, \lambda)$, where Σ, Q, $Q_0 \subseteq Q$, δ is as before and Δ is the output alphabet, $\lambda : Q \to \Delta$ the output function. The outputs of a Moore machine, defined by the function λ, are thus determined by the current state $q \in Q$ alone, rather than by input symbols. As before, δ extends to the domain of words as expected. For a deterministic Moore machine, we denote by λ also the function that applied to a word u yields the output in the state reached by u rather than the sequence of outputs.

3 Three-Valued LTL in the Untimed Setting

To overcome difficulties in defining an adequate boolean semantics for LTL on finite traces, we propose a 3-valued semantics. The intuition is as follows: in theory, we observe an infinite sequence w of some system. For a given formula φ, thus either $w \models \varphi$ or not. In practice, however, we can only observe a finite prefix u of w. Consequently, we let the semantics of u and φ be true, if $uw' \models \varphi$ for every possible future extension w'. On the other hand, if uw' is not a model of φ for all possible infinite continuations w' of u, we define the semantics of u and φ as false. In the remaining case, the truth value of uw' and φ depends on w'. Thus, we define the semantics of u with respect to φ to be *inconclusive*, denoted by ?, to signal that u itself is not sufficient to determine how φ will evaluate in any possible future which is prefixed with u.

Formally, we define our 3-valued semantics in terms of LTL$_3$ over the set of truth values $\mathbb{B}_3 = \{\bot, ?, \top\}$ as follows:

Definition 1 (3-valued semantics of LTL). *Let $u \in \Sigma^*$ denote a finite trace. The* truth value *of a LTL$_3$ formula φ w. r. t. u, denoted by $[u \models \varphi]$, is an element of $\mathbb{B}_3$ and defined as follows:*

$$[u \models \varphi] = \begin{cases} \top & \textit{if } \forall \sigma \in \Sigma^\omega : u\sigma \models \varphi \\ \bot & \textit{if } \forall \sigma \in \Sigma^\omega : u\sigma \not\models \varphi \\ ? & \textit{otherwise.} \end{cases}$$

Now, we develop an automata-based monitor procedure for LTL$_3$. More specifically, for a given formula $\varphi \in$ LTL$_3$, we construct a finite Moore machine, $\bar{\mathcal{A}}^\varphi$ that reads finite traces $u \in \Sigma^*$ and outputs $[u \models \varphi]$, thus a value in $\mathbb{B}_3$.

For a NBA $\mathcal{A}$, we denote by $\mathcal{A}(q)$ the NBA that coincides with $\mathcal{A}$ except for Q_0, which is defined as $Q_0 = \{q\}$. Fix $\varphi \in$ LTL for the rest of this section and let $\mathcal{A}^\varphi$ denote the NBA, which accepts all models of φ, and let $\mathcal{A}^{\neg\varphi}$ denote the NBA, which accepts all counter examples of φ. The corresponding construction is standard and explained, for example in [26]. For these automata, we observe:

Lemma 1. *Let $\mathcal{A}^\varphi = (\Sigma, Q^\varphi, Q_0^\varphi, \delta^\varphi, F^\varphi)$ denote the NBA such that $\mathcal{L}(\mathcal{A}^\varphi) = \mathcal{L}(\varphi)$. For $u \in \Sigma^*$, let $\delta(Q_0^\varphi, u) = \{q_1, \ldots, q_l\}$. Then*

$$[u \models \varphi] \neq \bot \textit{ iff } \exists q \in \{q_1, \ldots, q_l\} \textit{ such that } \mathcal{L}(\mathcal{A}^\varphi(q)) \neq \emptyset.$$

Lemma 2. *Let $\mathcal{A}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, Q_0^{\neg\varphi}, \delta^{\neg\varphi}, F^{\neg\varphi})$ denote the NBA such that $\mathcal{L}(\mathcal{A}^{\neg\varphi}) = \mathcal{L}(\neg\varphi)$. For $u \in \Sigma^*$, let $\delta(Q_0^{\neg\varphi}, u) = \{q_1, \ldots, q_l\}$. Then*

$$[u \models \varphi] \neq \top \textit{ iff } \exists q \in \{q_1, \ldots, q_l\} \textit{ such that } \mathcal{L}(\mathcal{A}^{\neg\varphi}(q)) \neq \emptyset.$$

Correctness of the first lemma follows directly from the definition of acceptance for Büchi automata and the second lemma rephrases the first one by substituting $\neg\varphi$ for φ.

For $\mathcal{A}^\varphi$ and $\mathcal{A}^{\neg\varphi}$, we now define a function $\mathcal{F}^\varphi : Q^\varphi \to \mathbb{B}$ respectively $\mathcal{F}^{\neg\varphi} : Q^{\neg\varphi} \to \mathbb{B}$ (where $\mathbb{B} = \{\top, \bot\}$), assigning to each state q whether the language of the respective automaton starting in state q is not empty. Thus, if $\mathcal{F}^\varphi(q) = \top$ holds, then the automaton $\mathcal{A}^\varphi$ starting at state q accepts a non-empty language and each finite prefix u leading to state q can be expanded in order to satisfy φ. Using $\mathcal{F}^\varphi$ and $\mathcal{F}^{\neg\varphi}$, we define two NFAs $\hat{\mathcal{A}}^\varphi = (\Sigma, Q^\varphi, Q_0^\varphi, \delta^\varphi, \hat{F}^\varphi)$ and $\hat{\mathcal{A}}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, Q_0^{\neg\varphi}, \delta^{\neg\varphi}, \hat{F}^{\neg\varphi})$ where $\hat{F}^\varphi = \{q \in Q^\varphi \mid \mathcal{F}^\varphi(q) = \top\}$ and $\hat{F}^{\neg\varphi} = \{q \in Q^{\neg\varphi} \mid \mathcal{F}^{\neg\varphi}(q) = \top\}$.

$\hat{\mathcal{A}}^\varphi$, resp. $\hat{\mathcal{A}}^{\neg\varphi}$, accept the finite traces u for which $[u \models \varphi]$ evaluates to $\neq \bot$ and, respectively, $\neq \top$.

Lemma 3. *Using the notation as before, we have for all $u \in \Sigma^*$:*

$$u \in \mathcal{L}(\hat{\mathcal{A}}^\varphi) \textit{ iff } [u \models \varphi] \neq \bot \quad \textit{and} \quad u \in \mathcal{L}(\hat{\mathcal{A}}^{\neg\varphi}) \textit{ iff } [u \models \varphi] \neq \top$$

Therefore, we can evaluate $[u \models \varphi]$ according to Lemma 3 as follows.

Lemma 4. *With the notation as before, we have $[u \models \varphi] = \top$ if $u \notin \mathcal{L}(\hat{\mathcal{A}}^{\neg\varphi})$, $[u \models \varphi] = \bot$ if $u \notin \mathcal{L}(\hat{\mathcal{A}}^\varphi)$, and $[u \models \varphi] = ?$ if $u \in \mathcal{L}(\hat{\mathcal{A}}^\varphi)$ and $u \in \mathcal{L}(\hat{\mathcal{A}}^{\neg\varphi})$.*

The lemma yields a simple procedure to evaluate the semantics of φ for a given finite trace u: we evaluate both $u \in \mathcal{L}(\hat{\mathcal{A}}^{\neg\varphi})$ and $u \in \mathcal{L}(\hat{\mathcal{A}}^\varphi)$ and use Lemma 4 to determine $[u \models \varphi]$. As a final step, we now define a (deterministic) FSM $\bar{\mathcal{A}}^\varphi$ that outputs for each finite string u its associated 3-valued semantical evaluation with respect to some LTL-formula φ.

Let $\tilde{\mathcal{A}}^\varphi$ and $\tilde{\mathcal{A}}^{\neg\varphi}$ be the deterministic versions of $\hat{\mathcal{A}}^\varphi$ and $\hat{\mathcal{A}}^{\neg\varphi}$, which can be computed in the standard manner by power-set construction. Now, we define the FSM in question as a product of $\tilde{\mathcal{A}}^\varphi$ and $\tilde{\mathcal{A}}^{\neg\varphi}$:

Definition 2 (Monitor $\bar{\mathcal{A}}^\varphi$ for a LTL-formula φ). *Let $\tilde{\mathcal{A}}^\varphi = (\Sigma, Q^\varphi, \{q_0^\varphi\}, \delta^\varphi, \tilde{F}^\varphi)$ and $\tilde{\mathcal{A}}^{\neg\varphi} = (\Sigma, Q^{\neg\varphi}, \{q_0^{\neg\varphi}\}, \delta^{\neg\varphi}, \tilde{F}^{\neg\varphi})$ be the DFAs which correspond to the two NFAs $\hat{\mathcal{A}}^\varphi$ and $\hat{\mathcal{A}}^{\neg\varphi}$ as defined for Lemma 3. Then we define the* monitor *$\bar{\mathcal{A}}^\varphi = \tilde{\mathcal{A}}^\varphi \times \tilde{\mathcal{A}}^{\neg\varphi}$ as the FSM $(\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$, where $\bar{Q} = Q^\varphi \times Q^{\neg\varphi}$, $\bar{q}_0 = (q_0^\varphi, q_0^{\neg\varphi})$, $\bar{\delta}((q, q'), a) = (\delta^\varphi(q, a), \delta^{\neg\varphi}(q', a))$, and $\bar{\lambda} : \bar{Q} \to \mathbb{B}_3$ is defined by*

$$\bar{\lambda}((q, q')) = \begin{cases} \top & \textit{if } q' \notin \tilde{F}^{\neg\varphi} \\ \bot & \textit{if } q \notin \tilde{F}^\varphi \\ ? & \textit{if } q \in \tilde{F}^\varphi \textit{ and } q' \in \tilde{F}^{\neg\varphi}. \end{cases}$$

$$\begin{tikzcd}
\text{Input} & \text{(1) Formula} & \text{(2) NBA} & \text{(3) Emptiness per state} & \text{(4) NFA} & \text{(5) DFA} & \text{(6) FSM} \\
 & \varphi \arrow[r] & \mathcal{A}^{\varphi} \arrow[r] & \mathcal{F}^{\varphi} \arrow[r] & \hat{\mathcal{A}}^{\varphi} \arrow[r] & \tilde{\mathcal{A}}^{\varphi} \arrow[dr] & \\
\varphi \arrow[ur] \arrow[dr] & & & & & & \bar{\mathcal{A}} \\
 & \neg\varphi \arrow[r] & \mathcal{A}^{\neg\varphi} \arrow[r] & \mathcal{F}^{\neg\varphi} \arrow[r] & \hat{\mathcal{A}}^{\neg\varphi} \arrow[r] & \tilde{\mathcal{A}}^{\neg\varphi} \arrow[ur] &
\end{tikzcd}$$

Fig. 1. The procedure for getting $[u \models \varphi]$ for a given φ

We sum up our entire construction in Fig. 1 and conclude by formulating the correctness theorem.

Theorem 1. *Let $\varphi \in$ LTL$_3$ and let $\bar{\mathcal{A}}^{\varphi} = (\Sigma, \bar{Q}, \bar{q}_0, \bar{\delta}, \bar{\lambda})$ be the corresponding monitor. Then, for all $u \in \Sigma^*$ the following holds: $[u \models \varphi] = \bar{\lambda}(\bar{\delta}(\bar{q}_0, u))$.*

Complexity. Consider Fig. 1: Given φ, step 1 requires us to replicate φ and to negate it, i.e., it is linear in the original size. Step 2, the construction of the NBAs, causes an exponential blow-up in the worst-case. Steps 3 and 4, leading to $\hat{\mathcal{A}}^{\varphi}$ and $\hat{\mathcal{A}}^{\neg\varphi}$, do not change the size of the original automata. Then, computing the deterministic automata of step 5, might again require an exponential blow-up in size. In total the FSM of step 6 will have double exponential size with respect to $|\varphi|$.

While the size of the final FSM is in $O(2^{2^n})$ which sounds a lot, standard minimisation algorithms for FSMs can be used to derive an *optimal* deterministic monitor w. r. t. the number of states. Optimality implies that any other method, in the worst case, has the same complexity. Better complexity results in other approaches are either due to using a restricted fragment of LTL or otherwise imply that the chosen temporal operators might not limit the expressive power of LTL but sometimes impose long formulas for encoding the desired behaviour.

That said, we have implemented the determinisation of NFAs and the product for obtaining $\bar{\mathcal{A}}$ (steps 4–6) in an *on-the-fly* fashion. This technique is well known for example in compiler construction [1]. Our examples confirm huge savings in memory consumption.

4 Three-Valued LTL in the Timed Setting—TLTL

In this part, we extend the approach developed in the preceding section to the timed setting. Thus, the goal is to dynamically check real-time specifications formulated in a timed temporal logic. We use timed LTL (TLTL for short), a logic introduced in [21], in the form presented in [22]. The language expressible by a TLTL formula can be defined by *event-clock automata* [4], a subclass of *timed automata*. It was shown in [11] that TLTL corresponds exactly to the class of languages definable in first-order logic interpreted over timed words. Thus, it can be considered as the natural counterpart of LTL for the timed setting. Given the translation to event-clock automata in the literature [22], we base our timed runtime verification approach on TLTL and event-clock automata.

4.1 Preliminaries

Let us fix an alphabet Σ of actions for the rest of this section. In the timed setting, every symbol $a \in \Sigma$ is associated with an *event-recording clock*, x_a, and an *event-predicting* *clock*, y_a. An (infinite) *timed word* w over the alphabet Σ is an (infinite) sequence of *timed events* $(a_0, t_0)(a_1, t_1)\ldots$ consisting of symbols $a_i \in \Sigma$, and non-negative numbers $t_i \in \mathbb{R}^{\geq 0}$, such that for each $i \in \mathbb{N}$, $t_i < t_{i+1}$ (*strict monotonicity*), and for all $t \in \mathbb{R}^{\geq 0}$ there is an $i \in \mathbb{N}$ such that $t_i > t$ (*progress*). Furthermore, for w as above, we call its sequence of actions (the projection to the first component) the *untimed word* of w, denoted by $ut(w)$.

To simplify notation, we abbreviate $(\Sigma \times \mathbb{R}^{\geq 0})$ by $T\Sigma$. Thus, a finite timed word is an element of $T\Sigma^*$ and the domain of infinite timed words is denoted by $T\Sigma^\omega$. Given an (infinite) timed word w, the value of the event-recording clock variable x_a at position i of w equals $t_i - t_j$, where j represents the last position preceding i such that $a_j = a$. If no such position exists, then the value of x_a remains undefined, denoted by $\perp$. The event-predicting clock variable y_a at position i equals $t_k - t_i$, where k represents the next position after i such that $a_k = a$. If no such position exists, again, the variable remains undefined. The set of all event-clocks is denoted by $C_\Sigma = \{x_a, y_a \mid a \in \Sigma\}$. A *clock valuation function* over a timed word w, $\gamma_i : C_\Sigma \to \mathbb{R}^{\geq 0} \cup \{\perp\}$ assigns a positive real, or undefined value to each clock variable corresponding to position i. We abbreviate $\mathbb{R}^{\geq 0} \cup \{\perp\}$ by $T_\perp$.

A *clock constraint* compares a clock value to a natural number. Let $\Psi(C_\Sigma)$ denote the set of clock constraints over C_Σ. Formally, a clock constraint $\psi \in \Psi(C_\Sigma)$ is a conjunction of formulae of the form $z \bowtie c$, where $z \in C_\Sigma$, $\bowtie \in \{<, \leq, \geq, >\}$ and $c \in \mathbb{N}$. For clock constraint ψ and clock valuation function γ, we write $\gamma \models \psi$ to denote that w.r.t. γ, constraint ψ is fulfilled, where $\perp \bowtie c$ for $c \in \mathbb{N}$ and $\bowtie \in \{<, \leq, \geq, >\}$ does not hold and the remaining cases are defined in the expected manner.

4.2 Syntax and Semantics of TLTL$_3$

Let Σ be a finite set of actions. A set of formulas φ of TLTL is defined by the grammar

$$\varphi ::= \mathit{true} \mid a \mid \lhd_a \in I \mid \rhd_a \in I \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi\, U\, \varphi \mid X\varphi \quad (a \in \Sigma),$$

where $\lhd_a$ is the operator that measures the time elapsed since the last occurrence of a, and $\rhd_a$ the operator that predicts the next occurrence of a within a timed interval $I \in \mathcal{I}$. The set of intervals $\mathcal{I}$ contains intervals of the form (l, r), $[l, r)$, $(l, r]$, or $[l, r]$, where $l, r \in \mathbb{R}^{\geq 0} \cup \{\infty\}$. Without loss of generality, we assume $l < r$, except for $[l, r]$, and for intervals $(l, r]$, or $[l, r]$ that $r \neq \infty$. To simplify notation, we use $[\![$ and $]\!]$ for interval borders which can either be (or [, respectively),].

The semantics of TLTL formulae are defined inductively over infinite timed words $w \in T\Sigma^\omega$, where $w = (a_0, t_0)(a_1, t_1)\ldots$, and $i \in \mathbb{N}^{\geq 0}$ as follows: $w, i \models \mathit{true}$, $w, i \models \neg\varphi$ iff $w, i \not\models \varphi$, $w, i \models a$ iff $a_i = a$, $w, i \models \lhd_a \in I$ iff $\gamma_i(x_a) \in I$, $w, i \models \rhd_a \in I$ iff $\gamma_i(y_a) \in I$, $w, i \models \varphi_1 \vee \varphi_2$ iff $w, i \models \varphi_1$ or $w, i \models \varphi_2$, $w, i \models \varphi_1 U \varphi_2$ iff $\exists k \geq i$ with $w, k \models \varphi_2$ and $\forall l : (i \leq l < k \wedge w, l \models \varphi_1)$, $w, i \models X\varphi$ iff $w, i+1 \models \varphi$. Further, let $w \models \varphi$, iff $w, 0 \models \varphi$.

Analogously to the untimed case, we now define a 3-valued semantics for TLTL, from this point onwards denoted as TLTL$_3$, as follows:

Definition 3. *Let $u \in T\Sigma^*$ denote a finite timed trace. The truth value of a TLTL$_3$ formula φ w. r. t. u, denoted by $[u \models \varphi]$, is an element of $\mathbb{B}_3$ and defined as follows:*

$$[u \models \varphi] = \begin{cases} \top & \textit{if}\ \forall\sigma\ \textit{such that}\ u\sigma \in T\Sigma^{\omega}\ u\sigma \models \varphi \\ \bot & \textit{if}\ \forall\sigma\ \textit{such that}\ u\sigma \in T\Sigma^{\omega}\ u\sigma \not\models \varphi \\ ? & \textit{otherwise.} \end{cases}$$

4.3 Symbolic Runs of Event-Clock Automata

We first recall the definition of an event-clock automaton: Given a finite set of clocks, C_{Σ}, we define an event-clock automaton as a finite state automaton whose edges are annotated both with input symbols and with clock constraints as $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$, where Σ is a finite input alphabet, Q a finite set of states, $Q_0 \subseteq Q$ are initial states, $F \subseteq 2^{Q}$ is a set of accepting states (generalized Büchi acceptance condition) and $E \subseteq Q \times \Sigma \times \Psi(C_{\Sigma}) \times Q$ a set of transitions. An edge $e = (q, a, \psi, q')$ represents a transition from state q upon symbol a to q', where the clock constraint ψ then specifies when e is enabled. For an event-clock automaton $\mathcal{A}$, let $K_{\mathcal{A}}$ denote the biggest constant appearing in some constraint of $\mathcal{A}$; we write K when $\mathcal{A}$ is clear from the context.

A *timed run* θ of an automaton $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$ over a timed word $w \in T\Sigma^{\omega}$ starting in (q_0, γ_0) is an infinite sequence of state-valuation tuples and transitions $(q_0, \gamma_0) \overset{\alpha_1}{\rightarrow} (q_1, \gamma_1) \overset{\alpha_2}{\rightarrow} \ldots$ with $q_i \in Q$, and γ_i being the evaluation function assigning for every element from Σ the value of the recording and predicting event clocks corresponding to α_i, where $\alpha_i \in T\Sigma$ is a timed event of the form $(a_i \in \Sigma, t_i \in \mathbb{R}^{\geq 0})$, and for all $i \geq 1$ there is a transition in E of the form $(q_{i-1}, a_i, \psi, q_i)$ such that $\gamma_i \models \psi$. $\mathcal{A}_{ec}$ accepts θ, iff for each $F_i \in F$, a state $q \in F_i$ exists such that q occurs infinitely often in θ. γ_0 is *initial* (w.r.t. w) if $\gamma_0(x_a) = \bot$ and $\gamma_0(y_a) = t_i$ if $\alpha_i = (a, t_i)$ and for $j < i$ and $\alpha_j = (a_j, t_j)$, $a_j \neq a$, and $\gamma_0(y_a) = \bot$ if a does not occur in w. Then, the timed language accepted by $\mathcal{A}_{ec}$, denoted as $\mathcal{L}(\mathcal{A}_{ec})$, is the set of timed words for which an accepting run of $\mathcal{A}_{ec}$ exists starting in (q_0, γ_0), for some $q_0 \in Q_0$ and the initial γ_0.

For runtime verification predicting clock variables pose a problem, since information about the future occurrence of an action a is predicted, but this information is not available yet. We solve this problem by representing the value of some predicting clock variable *symbolically*. A *symbolic clock valuation function* $\Gamma : C_{\Sigma} \rightarrow T_{\bot} \cup \mathcal{I}$ assigns a positive real, or undefined value to each recording clock variable and an *interval* or undefined value to each predicting clock variable. The interval constrains the possible values of a predicting variable. To simplify notation, we identify $\Gamma(y_a) = (l, r)$ with the constraint $y_a > l \wedge y_a < r$ (and similarly for borders [and]).

For a symbolic clock evaluation Γ, we define the following three operations: time *elapse*, *reset*, and *conjunction*. Given an *elapsed* time $t \in \mathbb{R}^{\geq 0}$, $\Gamma' = \Gamma + t$, where $\Gamma'(x_a) = \Gamma(x_a) + t$ and for $\Gamma(y_a) = [\![l, r]\!]$, we set $\Gamma'(y_a) = [\![l \dot{-} t, r - t]\!]$, where $\dot{-}$ yields at least 0. If $r - t < 0$, then Γ' is invalid. Γ *reset* by action a, denoted by $\Gamma \downarrow a$, sets $x_a = 0$ and removes all constraints on y_a, and we set $\Gamma'(y_a) = [0, \infty)$ and $\Gamma'(z_b) = \Gamma(z_b)$ for all $b \neq a$. The *conjunction* of Γ with constraint ψ yields $\Gamma' = \Gamma \wedge \psi$, where each predicting clock y_a is combined with the constraints of ψ which involve y_a, i. e., for $a \in \Sigma$, $\Gamma'(y_a) = \Gamma(y_a) \wedge \bigwedge\{y_a \bowtie c \subseteq \psi\}$. We call Γ' invalid, if for some y_a, $\Gamma'(y_a)$ is not satisfiable. Furthermore, a transition $(q, a, \psi, q') \in E$ is *applicable* to a pair (q, Γ), if the constraints $x_b \bowtie c$ in ψ are satisfied by Γ, for all $b \in \Sigma$, and $0 \in \Gamma(y_a)$. If $(q, a, \psi, q') \in E$ is applicable, then the corresponding successor of (q, Γ) is (q', Γ'), where $\Gamma' = (\Gamma \downarrow a) \wedge \psi$.

A *symbolic timed run* Θ of an automaton $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$ over a timed word $w \in T\Sigma^\omega$ starting in (q_0, Γ_0) is an infinite sequence of state-symbolic-valuation tuples and transitions as follows: $(q_0, \Gamma_0) \overset{\alpha_1}{\to} (q_1, \Gamma_1) \overset{\alpha_2}{\to} \dots$ with $q_i \in Q$, and Γ_i being a symbolic valuation function, where for each $(q_{i-1}, \Gamma_{i-1}) \overset{(a_i,t_i)}{\to} (q_i, \Gamma_i)$, there exists some transition $(q_{i-1}, a_i, \psi, q_i)$ applicable to $(q_{i-1}, \Gamma_{i-1} + t_i)$ and (q_i, Γ_i) is the result of this application. The notion of acceptance for symbolic runs corresponds to that of runs, i. e., for each $F_i \in F$ there is some $q \in F_i$ occurring infinitely often. We call Γ_0 *initial* if for $a \in \Sigma$, $\Gamma_0(x_a) = \bot$ and $\Gamma_0(y_a) = [0, \infty)$.

Theorem 2. *Let $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$ be an event-clock automaton and $w \in T\Sigma^\omega$. Then, there is an accepting run on w starting in (q_0, γ_0) iff there is a symbolic accepting run on w starting in (q_0, Γ_0) for initial Γ_0.*

The important fact about the previous theorem is that γ_0 is dependent on w (since each predicting clock y_a has to be initialised to match the first occurrence of a), while Γ_0 is independent of w. Thus, symbolic runs are a suitable device for runtime verification.

4.4 A Monitor Procedure for TLTL$_3$

We can assume that for some property φ as well as its negation, an event-clock automaton is given, accepting precisely the models of φ respectively $\neg\varphi$ (see [22] for details). Looking at the scheme developed in the untimed setting, we are now tempted to check for every state q of the event-clock automaton, whether the language accepted from state q is empty. However, this would yield wrong conclusions, as can be seen in Fig. 2. While the language accepted in state 2 is non-empty and, despite, state 2 is reachable, the automaton does not accept any word when starting in state 0. The constraint when passing from 1 to 2 requires the clock x_a to be at least 2. This, however, prevents the loop in state 2 to be taken.

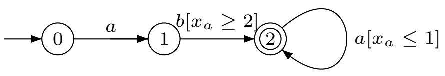


Fig. 2. Event-clock automaton

We therefore decided to work on the so-called region automaton (for alternatives see Remark 2 on page 270). Recall that K denotes the biggest constant occurring in some constraint of the event-clock automaton. Two clock valuations γ_1, γ_2 are in the same region, denoted by $\gamma_1 \equiv \gamma_2$ iff

- for all $z \in C_\Sigma$, $\gamma_1(z) = \bot$ iff $\gamma_2(z) = \bot$, and (*agreement on undefined*)
- for all $z \in C_\Sigma$, if $\gamma_1(z) \le K$ or $\gamma_2(z) \le K$, then $\lfloor \gamma_1(z) \rfloor = \lfloor \gamma_2(z) \rfloor$, and (*agreement on integral part*)
- for all $a \in \Sigma$, let $\langle \gamma(x_a) \rangle = \lceil x_a \rceil - \gamma(x_a)$ and $\langle \gamma(y_a) \rangle = \gamma(y_a) - \lfloor y_a \rfloor$. Then, for all $z_1, z_2 \in C_\Sigma$ with $\gamma_1(z_1) \le K$ and $\gamma_2(z_2) \le K$,
  - $\langle \gamma_1(z_1) \rangle = 0$ iff $\langle \gamma_2(z_1) \rangle = 0$
  - $\langle \gamma_1(z_1) \rangle \le \langle \gamma_1(z_2) \rangle$ iff $\langle \gamma_2(z_1) \rangle \le \langle \gamma_2(z_2) \rangle$. (*agreement on fraction's order*)

A *clock region* is an equivalence class of $\equiv$. Let $\mathcal{R}$ denote the set of all regions.

The key property of the region equivalence is *stability* [3]: given state s and two equivalent valuations γ_1 and γ_2, then (s', γ') is an a-successor of (s, γ_1) iff (s', γ'') is one of (s, γ_2) for suitable γ'' equivalent to γ'. Lifting this to infinite runs, we get:

Lemma 5. *Let $\mathcal{A}_{ec}$ be an event-clock automaton, q some state of $\mathcal{A}_{ec}$, and γ_1, γ_2 two valuations with $\gamma_1 \equiv \gamma_2$. Let $\bar{w} \in \Sigma^\omega$. Then, there exists an accepting run on some infinite timed word $w_1 \in T\Sigma^\omega$ with $ut(w_1) = \bar{w}$ starting in (q, γ_1) iff there exists an accepting run on some infinite timed word $w_2 \in T\Sigma^\omega$ with $ut(w_2) = \bar{w}$ starting in (q, γ_2).*

Note that the so-called *zone equivalence* [2] is not stable.

For completeness, we give the translation of an event-clock automaton to a region automaton, as presented in [22], whose states actually serve their purpose in our approach, because of the previous lemma.

A clock region κ_2 is a *time successor* of a clock region κ_1, denoted by $\kappa_2 \in TS(\kappa_1)$, iff for all $\gamma \in \kappa_1$ there is some $t \in \mathbb{R}^{\geq 0}$ such that $\gamma + t \in \kappa_2$. Here, $\gamma' = \gamma + t$ is defined as $\gamma'(x_a) = \gamma(x_a) + t$ and $\gamma'(y_a) = \gamma(y_a) - t$. To simplify notation, let us fix an event-clock automaton $\mathcal{A}_{ec} = (\Sigma, Q, Q_0, E, F)$. The region automaton of $\mathcal{A}_{ec}$ is the (generalized) Büchi automaton $R(\mathcal{A}_{ec}) = (\Sigma^r, Q^r, Q_0^r, E^r, F^r)$, where

- $Q^r = \{(l, \kappa, \zeta) \mid l \in Q, \kappa \in \mathcal{R}, \zeta \in \{t, d\}\}$ is the set of states
- $Q_0^r = \{(l, \kappa, \zeta) \in Q^r \mid l \in Q_0, \forall a \in \Sigma, \kappa(x_a) = \bot, \zeta = d\}$ is the set of initial states
- $\Sigma^r = \Sigma \cup \{\epsilon\}$
- $E^r = E_d^r \cup E_t^r$ is the union of untimed and timed transitions, where
  - $E_d^r = \{((l_1, \kappa_1, t), (l_2, \kappa_2, d), a) \mid (l_1, a, \psi, l_2) \in E$ and $\exists \kappa_3$ s. t. $\kappa_1 = \kappa_3[y_a := 0], \kappa_2 = \kappa_3[x_a := 0]$, and $\kappa_3 \models \psi\}$
  - $E_t^r = \{((l, \kappa_1, d), (l, \kappa_2, t), \epsilon) \mid \kappa_2 \in TS(\kappa_1)\}$
- $F^r = \{F_i^r \mid F_i \in F\} \cup \{F_{x_a} \mid \lhd_a \in I \in Sub(\varphi)\} \cup \{F_{y_a} \mid \rhd_a \in I \in Sub(\varphi)\}$,
  - where for $F_i \in F$, $F_i^r = \{(l, \kappa, \zeta) \mid l \in F_i\}$
  - $F_{x_a} = \{(l, \kappa, \zeta) \mid \forall \gamma \in \kappa\, \gamma(x_a) = 0 \vee \gamma(x_a) > c \vee \gamma(x_a) = \bot\}$
  - $F_{y_a} = \{(l, \kappa, \zeta) \mid \forall \gamma \in \kappa\, \gamma(y_a) = 0 \vee \gamma(y_a) = \bot\}$

Note that the region automaton as defined here is a Büchi automaton and thus, the accepted language is a sequence of (untimed) words over Σ. Thus, it is easy to compute for every state, whether the accepted (untimed) language is empty or not. For every state (l, κ, ζ) with a non-empty language, stability now guarantees that for each $\gamma \in \kappa$, there is some accepting run of the original event-clock automaton starting in (l, γ) for some timed word w. Dually, if the accepted language is empty, the underlying event-clock automaton has no accepting run starting in (l, γ) for any $\gamma \in \kappa$ and any w (Lemma 5).

We now describe a procedure that reads timed events and decides whether further events might yield an accepting run (satisfying the formula to check).

The procedure is based on the event-clock automaton as well as the region automaton. It follows the possible *symbolic* computations for the given input along the lines of the event-clock automaton. To decide, whether future events might contribute to an accepting run, the region automaton is consulted.

Let us fix an event-clock automaton $\mathcal{A}_{ec}$ and its region automaton $R(\mathcal{A}_{ec})$ for the moment. Let us consider the timed word $w = (a_0, t_0)(a_1, t_1) \cdots \in T\Sigma^\omega$. Recall that (a_0, t_0) actually means that the first action a_0 occurs at time t_0.

Let Γ_0 be the initial symbolic valuation of $\mathcal{A}_{ec}$ and l_0 one of the initial states of $\mathcal{A}_{ec}$. Now, for the first event (a_0, t_0), we compute the set of successors w.r.t. $\mathcal{A}_{ec}$. If this set is empty, the underlying formula is obviously violated. If not, each successor

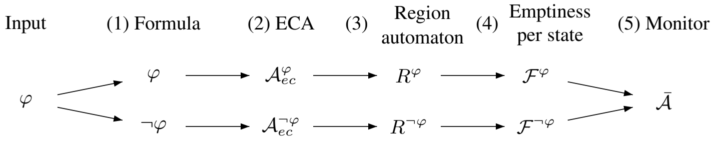


Fig. 3. The procedure for getting $[u \models \varphi]$ for a given $\varphi \in$ TLTL3

is a pair (l, Γ). Each (l, Γ) now corresponds to a set of states in the region automaton. If *and only if* for all of them the accepted language is empty, the underlying property is violated, which follows directly from Theorem 2 and Lemma 5. We continue with each successor state (l, Γ) for which a corresponding accepting state of $R(\mathcal{A}_{ec})$ exists, reading the input event.

Thus, the generated procedure keeps a set of possible state-symbolic valuation pairs that represent the possible current states of $\mathcal{A}_{ec}$ (giving credit to the non-deterministic nature of $\mathcal{A}_{ec}$). Furthermore, the transition table of $\mathcal{A}_{ec}$ and the states of $R(\mathcal{A}_{ec})$ enriched with emptiness per state information are stored as look-up tables.

Remark 1. To enhance the practical applicability of our approach, we adjust the procedure slightly: the formal framework described above requires the monitor to complain iff for some prefix $(a_0, t_0) \dots (a_i, t_i)$ no accepting run exists. In particular, it is assumed that "a watch is consulted only when some action occurs". But the time transitions yielding the subsequent regions in the region automaton actually (often) constrain the possible occurrence of some future event a. For each current valuation Γ corresponding to a set of regions, we check in $R(\mathcal{A})$ the possible accepting time successors and compute a maximal time bound before some event has to occur to reach an accepting state. Thus, in practice, we can set a timer interrupt, when such a bound exists, and decide for rejection, when a timeout occurs before a suitable action has been read.

The overall monitor procedure for TLTL3 is similar to the untimed case and summarised in Fig. 3. However, since we have to consider the region automaton (with emptiness per state information) together with the current clock valuation to compute the timed successor, we do not get an NFA neither can determinise to get a DFA (at least in a straightforward manner). We therefore propose for the overall monitor procedure to rely on $R(\mathcal{A}_{ec}^{\varphi})$ and $R(\mathcal{A}_{ec}^{\neg\varphi})$ in an on-the-fly manner, as described above.

Remark 2. We have used region automata to keep our presentation short and simple. The key property of our monitor construction, however, is *stability* of the region equivalence. Thus, our approach can be improved by taking a coarser stable partition of the underlying timed transition system instead of the region equivalence. Such partitions have been studied extensively in [25].

Complexity. Consider Fig. 3 and observe that step 1 is constant. The region automaton of $\mathcal{A}_{ec}^{\varphi}$ (resp. $\mathcal{A}_{ec}^{\neg\varphi}$) is exponential with respect to the length of the underlying formula φ as well as the largest constant K appearing in φ. Following the different paths for some prefix (due to the non-determinism of the region automaton) might cause further exponential blow-up in space, in the worst case.

5 Conclusions

The paper presented a monitor construction for (T)LTL formulae. For LTL, we have shown the construction to be optimal, in that no smaller deterministic finite state monitor accepting the same language as ours can be constructed. For both, LTL and TLTL, the construction reflects minimality, such that *true* or *false* is returned by the monitor as early as an observed behavioural trace allows. The latter is an implicit property of the constructed monitor and does not require additional analyses, or data structures besides the monitor itself.

We have already implemented the untimed setting and integrated the monitor generator within a larger logging and unit testing framework. Examples and an extended version of this paper including details of the implementation are available from `http://runtime.in.tum.de/`

References

1. A. Aho, R. Sethi, and J. Ullman. *Compilers: Principles and Techniques and Tools*. Addison-Wesley, 1986.
2. R. Alur. Timed automata. In *NATO-ASI 1998 Summer School on Verification of Digital and Hybrid Systems*, 1998.
3. R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2):183–235, 1994.
4. R. Alur, L. Fix, and T. A. Henzinger. Event-clock automata: A determinizable class of timed automata. *TCS*, 211(1-2):253–273, 1999.
5. A. Bauer, M. Leucker, and C. Schallhart. Model-based runtime analysis of distributed reactive systems. In *ASWEC'06*. IEEE, 2006.
6. P. Bouyer, F. Chevalier, and D. D'Souza. Fault diagnosis using timed automata. In *FoSSaCS*, LNCS 3441. Springer, 2005.
7. M. Broy, B. Jonsson, J.-P. Katoen, M. Leucker, and A. Pretschner, editors. *Model-based Testing of Reactive Systems*, LNCS 3472. Springer, 2005.
8. M. Chechik, B. Devereux, and A. Gurfinkel. Model-checking infinite state-space systems with fine-grained abstractions using spin. In *SPIN'01*, LNCS 2057.
9. S. Colin and L. Mariani. *Run-Time Verification*, chapter 18. LNCS 3472. [7], 2005.
10. M. d'Amorim and G. Rosu. Efficient monitoring of omega-languages. In *CAV'05*, LNCS 3576. Springer, 2005.
11. D. D'Souza. A logical characterisation of event clock automata. *Int. Journ. Found. Comp. Sci.*, 14(4):625–639, Aug. 2003.
12. C. Eisner, D. Fisman, J. Havlicek, Y. Lustig, A. McIsaac, and D. V. Campenhout. Reasoning with temporal logic on truncated paths. In *CAV'03*, LNCS 2725.
13. D. Giannakopoulou and K. Havelund. Automata-Based Verification of Temporal Properties on Running Programs. In *ASE'01*, IEEE, 2001.
14. J. Håkansson, B. Jonsson, and O. Lundqvist. Generating online test oracles from temporal logic specifications. *STTT*, 4(4):456–471, 2003.
15. K. Havelund and G. Rosu. Monitoring Java Programs with Java PathExplorer. *ENTCS*, 55(2), 2001.
16. K. Havelund and G. Rosu. Monitoring programs using rewriting. In *ASE'01*, IEEE, 2001.
17. K. Havelund and G. Rosu. Synthesizing Monitors for Safety Properties. In *TACAS'02*, 2002.
18. O. Kupferman and M. Y. Vardi. Model checking of safety properties. *FMSD*, 19(3):291–314, 2001.

19. O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *FORMATS/FTRTFT*, LNCS 3253. Springer, 2004.
20. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*. IEEE, 1977.
21. J.-F. Raskin and P.-Y. Schobbens. State clock logic: A decidable real-time logic. In O. Maler, editor, *HART*, LNCS 1201. Springer, 1997.
22. J.-F. Raskin and P.-Y. Schobbens. The logic of event clocks—decidability, complexity and expressiveness. *JALC*, 4(3):247–286, 1999.
23. V. Stolz and E. Bodden. Temporal Assertions using AspectJ. In *Fifth Workshop on Runtime Verification (RV'05)*. ENTCS. to appear.
24. S. Tripakis. Fault diagnosis for timed automata. In W. Damm and E.-R. Olderog, editors, *FTRTFT*, LNCS 2469. Springer, 2002.
25. S. Tripakis and S. Yovine. Analysis of timed systems using time-abstracting bisimulations. *FMSD*, 18(1):25–68, 2001.
26. M. Y. Vardi. *An Automata-Theoretic Approach to Linear Temporal Logic*, LNCS 1043. Springer, 1996.

A Proof System for the Linear Time μ-Calculus

Christian Dax[1], Martin Hofmann[2], and Martin Lange[2]

[1] Department of Computer Science, ETH Zürich
[2] Institut für Informatik, LMU München

Abstract. The linear time μ-calculus extends LTL with arbitrary least and greatest fixpoint operators. This gives it the power to express all ω-regular languages, i.e. strictly more than LTL. The validity problem is PSPACE-complete for both LTL and the linear time μ-calculus. In practice it is more difficult for the latter because of nestings of fixpoint operators and variables with several occurrences.

We present a simple sound and complete infinitary proof system for the linear time μ-calculus and then present two decision procedures for provability in the system, hence validity of formulas. One uses nondeterministic Büchi automata, the other one a generalisation of size-change termination analysis (SCT) known from functional programming.

The main novelties of this paper are the connection with SCT and the fact that both decision procedures have a better asymptotic complexity than earlier ones and have been implemented.

1 Introduction

The linear time μ-calculus (L_μ^{lin}) [1,14] extends Pnueli's *Linear Time Temporal Logic* (LTL) with extremal fixpoints quantifiers. This increases its expressive power: L_μ^{lin} captures exactly the ω-regular languages, while the class of LTL-definable properties is only that of star-free ω-languages. L_μ^{lin} can also be seen as the modal μ-calculus which is only interpreted over infinite linear time structures, i.e. Kripke structures in which every state has exactly one successor.

The main decision problems for LTL and L_μ^{lin} have the same complexity: model checking, satisfiability and validity are all PSPACE-complete for both logics [11,14]. By *model checking* we denote, as usual, the problem to decide whether all paths of a given Kripke structure satisfy a given specification. Note that these three problems are all interreducible for linear time logics. For instance, validity is the same as model checking in a *universal Kripke structure* that has the shape of a full clique; model checking can be reduced to validity checking by modeling the given structure in a formula which is linear in the size of the structure, etc. Since these reductions do not interfere with the main difficulty in each decision problem – to find infinite regenerations of least or greatest fixpoint – we will simply refer to *decision problems*. Here we focus on the validity problem but stress the point that this approach is applicable to the other problems without major alterations, too.

The presence of nested and possibly alternating fixpoint constructs makes L_μ^{lin}'s decision problems much harder in practice than those of LTL. The fact

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 273–284, 2006.

that LTL formulas only contain very simple unnested fixpoints is certainly one of the reasons for LTL being well supported by successfully working tools like SPIN and NUSMV, etc.

Some decision procedures for L_μ^{lin} have been presented so far. Vardi [14] uses nondeterministic Büchi automata to decide an extension of L_μ^{lin} with temporal past operators. The time complexity of his algorithm is $2^{O(n^4)}$ where n is the size of the input formula. Stirling and Walker subsequently gave a tableau characterisation for L_μ^{lin}'s decision problems but were not concerned with complexity issues.

Bradfield, Esparza and Mader defined tableaux with simpler termination conditions. Their algorithm runs in time $2^{O(n^2 \log n)}$ but this appeals to general complexity theorems about nondeterministic space vs. deterministic time. Hence, their result is of theoretical rather than – as they say – practical use. The same holds for Kaivola's procedure [4] which runs in time $2^{O(n^2 \log n)}$ when transformed into a deterministic procedure. We remark that it was designed to be nondeterministic in the first place – the user is supposed to provide Hintikka structures manually. To the best of our knowledge, none of these existing suggestions to solve L_μ^{lin}'s decision problems have ever seen any serious attempt to be put into practice.

Here we present a simple proof system for L_μ^{lin}. A proof is an infinite tree in which each branch satisfies an additional *global* condition concerning the existence of *threads* – similar to the internal paths of [2]. Our proof system and in particular the characterisation of valid proof branches is related to the notion of pre-models and models in Streett and Emerson's work on deciding the modal μ-calculus [13], adapted to L_μ^{lin} by Vardi [14]. Indeed, a formula is invalid iff its negation is satisfiable and in this case the offending path in the generic pre-proof amounts to a model in their sense when we negate all formulas and understand a sequent as the conjunction of its formulas whereas any infinite path in a pre-proof can be extended to a pre-model.

There are some subtle differences though. States of a pre-model are always maximally consistent sets of formulas (Hintikka sets) whereas our proofs contain arbitrary sequents. Second, by considering the whole proof tree rather than individual paths in isolation the need for the perhaps mysterious concept of choice functions disappears. Of course they come back in Section 4 where we show that a simple nondeterministic parity (or Büchi) automaton (NPA/NBA) is able to accept all valid paths in a proof. They return in the form of a condensed description of rule instances fed to the NPA in addition to the sequents. We claim though that the concept is more naturally explained by arguing that the automaton must check every path in the pre-proof.

We present two different approaches to decide validity. The first one reduces this to the inclusion problem for nondeterministic Büchi automata. Depending on which complementation procedure is used we obtain an algorithm that runs in time $2^{O(n^2 \log n)}$ for example. This is easily implementable since it does not use any theorems from complexity theory. Alternatively, there is a procedure running in time $2^{O(n^4)}$ that can be implemented symbolically.

The second approach is an iterative algorithm inspired by the size-change termination (SCT) method introduced by Jones et al. [8] in the context of termination analysis. There is, effectively, a fundamental connection between termination of functional programs and decision problems for ω-automata which we will elaborate elsewhere. Here we adapt and substantially generalise the SCT method in an ad hoc fashion to our situation at hand. Validity then reduces to the problem of finding an idempotent morphism satisfying a certain property in a category generated by a finite number of morphisms. Rule applications in the proof system are regarded as morphisms with successive applications as morphism composition. Systematically exploring the set of morphisms can be done in time $2^{O(n^3)}$ but is in practice better than the automata-theoretic method as some experimental results suggest.

2 Preliminaries

Let $\mathcal{P} = \{p, q, \ldots\}$ be a set of atomic propositions, and $\mathcal{V} = \{X, Y, \ldots\}$ an infinite set of monadic second-order variables. Formulas of the linear time μ-calculus L_μ^{lin} in positive normal form are given by the following grammar.

$$\varphi \;::=\; q \mid \neg q \mid X \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \mu X.\varphi \mid \nu X.\varphi$$

where $q \in \mathcal{P}$, and $X \in \mathcal{V}$. We will write σ for either μ or ν and use $l, \bar{l}$ etc. to denote literals $q, \neg q$ and their complements. We assume the reader to be familiar with the standard notions of syntactic subformulas $\mathit{Sub}(\varphi)$, free variables, well-named formulas, substitution $\varphi[\psi/X]$ of all occurrences of a variable, etc.

The Fischer-Ladner closure $\mathit{FL}(\varphi_0)$ of a L_μ^{lin}-formula φ_0 is the least set of formulas that contains φ_0 and satisfies: if $\varphi \in \mathit{FL}(\varphi_0)$ and

- $\varphi = \psi_1 \vee \psi_2$ or $\psi = \psi_1 \wedge \psi_2$ then $\{\psi_1, \psi_2\} \subseteq \mathit{FL}(\varphi)$;
- $\varphi = \bigcirc\psi$ then $\psi \in \mathit{FL}(\varphi)$;
- $\varphi = \sigma X.\psi$ then $\psi[\sigma X.\psi/X] \in \mathit{FL}(\varphi)$.

Define $|\varphi_0| := |\mathit{FL}(\varphi_0)|$. Note that $|\mathit{FL}(\varphi_0)|$ is bounded by the syntactical length of φ_0.

A *linear time structure* over $\mathcal{P}$ is a function $\mathcal{K} : \mathbb{N} \to 2^{\mathcal{P}}$ or, equally, an ω-word over the alphabet $2^{\mathcal{P}}$. The semantics of a L_μ^{lin}-formula φ, relative to $\mathcal{K}$ and an environment $\rho : \mathcal{V} \to 2^{\mathbb{N}}$ is a subset of $\mathbb{N}$, inductively defined using the Knaster-Tarski-Theorem.

$$\begin{array}{llll}
[\![q]\!]_\rho^\mathcal{K} & := \{n \in \mathbb{N} \mid q \in \mathcal{K}(n)\} & [\![X]\!]_\rho^\mathcal{K} & := \rho(X) \\
[\![\neg q]\!]_\rho^\mathcal{K} & := \{n \in \mathbb{N} \mid q \notin \mathcal{K}(n)\} & [\![\bigcirc\varphi]\!]_\rho^\mathcal{K} & := \{n \in \mathbb{N} \mid n+1 \in [\![\varphi]\!]_\rho^\mathcal{K}\} \\
[\![\mu X.\varphi]\!]_\rho^\mathcal{K} & := \bigcap\{T \subseteq \mathbb{N} \mid [\![\varphi]\!]_{\rho[X \mapsto T]}^\mathcal{K} \subseteq T\} & [\![\varphi \vee \psi]\!]_\rho^\mathcal{K} & := [\![\varphi]\!]_\rho^\mathcal{K} \cup [\![\psi]\!]_\rho^\mathcal{K} \\
[\![\nu X.\varphi]\!]_\rho^\mathcal{K} & := \bigcup\{T \subseteq \mathbb{N} \mid T \subseteq [\![\varphi]\!]_{\rho[X \mapsto T]}^\mathcal{K}\} & [\![\varphi \wedge \psi]\!]_\rho^\mathcal{K} & := [\![\varphi]\!]_\rho^\mathcal{K} \cap [\![\psi]\!]_\rho^\mathcal{K}
\end{array}$$

We write $\mathcal{K}, i \models_\rho \varphi$ if $i \in [\![\varphi]\!]_\rho^\mathcal{K}$, and $\mathcal{K} \models_\rho \varphi$ if $0 \in [\![\varphi]\!]_\rho^\mathcal{K}$. If φ is closed we may also drop ρ.

By deMorgan's laws and duality of μ and ν, negation $\neg$ – and then of course $\rightarrow$ and $\leftrightarrow$ – can be defined in L_μ^{lin}.

A formula φ is *valid*, written $\models \varphi$, if for all linear time structures $\mathcal{K}$, and all environments ρ: $\mathcal{K} \models_\rho \varphi$ holds. Two formulas φ and ψ are equivalent, $\varphi \equiv \psi$, if for all ρ, and all $\mathcal{K}$ we have $\mathcal{K} \models_\rho \varphi$ iff $\mathcal{K} \models_\rho \psi$.

A formula is guarded if every occurrence of a variable X is in the scope of a $\bigcirc$-operator under its quantifier μ or ν. Every L_μ^{lin} formula can be transformed into guarded form.

Approximants of a fixpoint formula $\nu X.\varphi$ are defined in the usual way: $\nu^0 X.\varphi := \mathtt{tt}$, $\nu^{k+1} X.\varphi := \varphi[\sigma^k X.\varphi/X]$, and $\nu^\omega X.\varphi := \bigwedge_{k\in\mathbb{N}} \nu^k X.\varphi$. The next result about approximants uses the fact that the semantics of a L_μ^{lin} formula is a monotone and continuous function (for infinite unions of directed sets) of type $2^{\mathbb{N}} \rightarrow 2^{\mathbb{N}}$ in each variable, c.f. [3].

Lemma 1. *For all linear time structures $\mathcal{K}$, all $i \in \mathbb{N}$, all environments ρ, and all $\varphi(X)$ we have: $\mathcal{K}, i \not\models_\rho \nu X.\varphi$ iff there is a $k \in \mathbb{N}$ s.t. $\mathcal{K}, i \not\models_\rho \nu^k X.\varphi$.*

3 A Proof System for the Linear Time μ-Calculus

Let φ_0 be fixed. A *sequent* is a subset Γ of $FL(\varphi_0)$. Semantically, a sequent stands for the disjunction of its members; the empty sequent is always false. We extend satisfaction by structures and validity to sequents accordingly.

A *pre-proof* for φ_0 is a possibly infinite tree whose nodes are labeled with sequents, whose root is labeled with $\vdash \varphi_0$ and which is built according to the following proof rules, later referred to as $(\vee)$, $(\wedge)$, (σ), and $(\bigcirc)$. We write $\bigcirc\Gamma$ to abbreviate $\bigcirc\gamma_1, \ldots, \bigcirc\gamma_n$ if $\Gamma = \gamma_1, \ldots, \gamma_n$.

$$\frac{\vdash \varphi, \psi, \Gamma}{\vdash \varphi \vee \psi, \Gamma} \qquad \frac{\vdash \varphi, \Gamma \qquad \vdash \psi, \Gamma}{\vdash \varphi \wedge \psi, \Gamma} \qquad \frac{\vdash \varphi[\sigma X.\varphi/X], \Gamma}{\vdash \sigma X.\varphi, \Gamma} \qquad \frac{\vdash \Gamma}{\vdash \bigcirc\Gamma, \Delta}$$

A *principal formula* in a rule application is a formula that gets transformed by this rule, e.g. $\varphi \vee \psi$ in rule $(\vee)$. Note that rule $(\bigcirc)$ can have several principal formulas.

For all sequents Γ, Δ and all rules r occurring in a pre-proof for φ_0, s.t. Γ is the conclusion of r and Δ is a premiss of r we define the *connection relation* $Con_r(\Gamma, \Delta) \subseteq FL(\varphi_0) \times FL(\varphi_0)$ as follows.

$$(\varphi, \psi) \in Con_r(\Gamma, \Delta) \quad \text{iff} \quad \begin{array}{l} \text{either } r \text{ does not transform } \varphi \text{ and } \varphi = \psi \\ \text{or } \psi \text{ results from } \varphi \text{ in the application of } r \end{array}$$

We drop the index r if the actual rule is irrelevant. Let $\pi = \Gamma_0, \Gamma_1, \ldots$ be an infinite branch in a pre-proof for φ_0 resulting from the rule applications $r_0, r_1, \ldots$. A *thread* in π is a sequence of formulas $\varphi_0, \varphi_1, \ldots$ s.t. for all $i \in \mathbb{N}$: $(\varphi_i, \varphi_{i+1}) \in Con_{r_i}(\Gamma_i, \Gamma_{i+1})$ holds. Such a thread is called a *ν-thread* if there is a $\nu X.\psi \in FL(\varphi_0)$ s.t. $\varphi_i = \nu X.\psi$ for infinitely many $i \in \mathbb{N}$, and for all $\mu Y.\psi'$ s.t. $\nu X.\psi \in Sub(\mu Y.\psi')$: there are only finitely many $i \in \mathbb{N}$ s.t. $\varphi_i = \mu Y.\psi'$. A μ-thread is defined accordingly.

The following facts about threads are not hard to see.

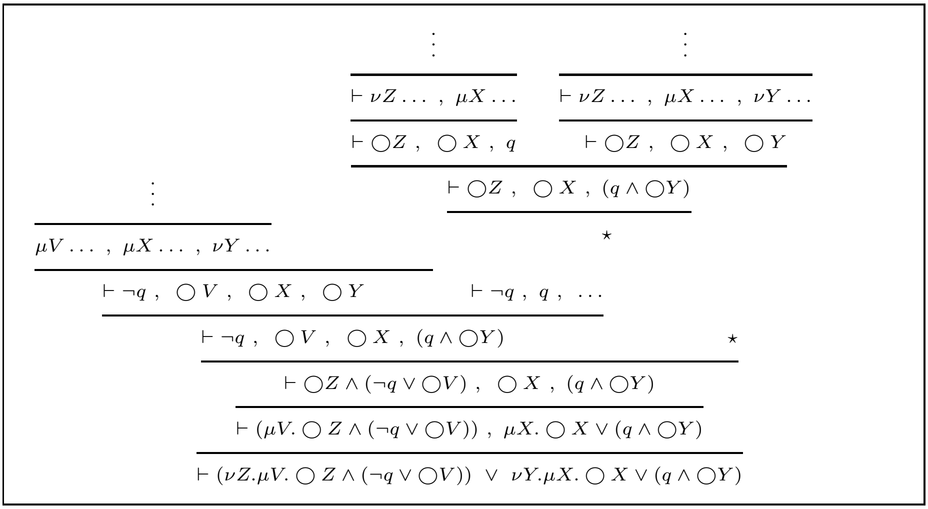

$\vdash \nu Z \ldots , \mu X \ldots$
$\vdash \nu Z \ldots , \mu X \ldots , \nu Y \ldots$
$\vdash \bigcirc Z , \bigcirc X , q$
$\vdash \bigcirc Z , \bigcirc X , \bigcirc Y$
$\vdash \bigcirc Z , \bigcirc X , (q \wedge \bigcirc Y)$
$\mu V \ldots , \mu X \ldots , \nu Y \ldots$
$\star$
$\vdash \neg q , \bigcirc V , \bigcirc X , \bigcirc Y$
$\vdash \neg q , q , \ldots$
$\vdash \neg q , \bigcirc V , \bigcirc X , (q \wedge \bigcirc Y)$
$\star$
$\vdash \bigcirc Z \wedge (\neg q \vee \bigcirc V) , \bigcirc X , (q \wedge \bigcirc Y)$
$\vdash (\mu V. \bigcirc Z \wedge (\neg q \vee \bigcirc V)) , \mu X. \bigcirc X \vee (q \wedge \bigcirc Y)$
$\vdash (\nu Z.\mu V. \bigcirc Z \wedge (\neg q \vee \bigcirc V)) \vee \nu Y.\mu X. \bigcirc X \vee (q \wedge \bigcirc Y)$

Fig. 1. Example of a proof

1. If a $\sigma X.\psi$ and a $\sigma' X'.\psi'$ occur infinitely often in a thread then $\sigma X.\psi \in Sub(\sigma' X'.\psi')$ or vice-versa.
2. Every thread is either a ν-thread or a μ-thread.

A *proof* for φ_0 is a pre-proof s.t. every finite branch ends in a sequent $l, \bar{l}, \Gamma$, and every infinite branch contains a ν-thread. We also write $\vdash \varphi_0$ to indicate that there is a proof for φ_0.

Example 1. Consider the quantifier swapping theorem

$$\models \ \big(\mu Z.\nu V. \bigcirc Z \vee (q \wedge \bigcirc V)\big) \ \rightarrow \ \big(\nu Y.\mu X. \bigcirc X \vee (q \wedge \bigcirc Y)\big)$$

This can be shown to be valid using principles from fixpoint theory. It is also intuitively valid: the premiss of the implication expresses "after some point, q always holds" and the conclusion says "q holds infinitely often".

In positive normal form this is written as $\varphi = (\nu Z.\mu V. \bigcirc Z \wedge (\neg q \vee \bigcirc V)) \vee (\nu Y.\mu X. \bigcirc X \vee (q \wedge \bigcirc Y))$. A proof for φ is sketched in Fig. 1. In order to save space, not all rule applications are listed explicitly and a variable is used to denote its unique fixpoint formula. On each infinite branch of this proof, either $\nu Z. \ldots$ or $\nu X. \ldots$ can be followed along a thread.

Theorem 1. *For all closed and guarded $\varphi \in L_\mu^{\text{lin}}$: if $\models \varphi$ then $\vdash \varphi$.*

Proof. Suppose $\models \varphi$. Let us replace $(\bigcirc)$ by the following restriction.

$$\frac{\vdash \Gamma}{\vdash \bigcirc \Gamma, l_1, \ldots, l_k} \ \nexists i, j : l_i = \overline{l_j}$$

Now all rules preserve and reflect validity. Therefore, systematic backwards application of the rules leads to a pre-proof P of φ comprising valid sequents only.

We claim that P is a proof. Note that guardedness means that all fixpoints must have been unfolded prior to an application of restricted $(\bigcirc)$ so no "round-robin" policy or similar is needed in the construction of P.

Take any infinite branch $\pi = \Delta_0, \Delta_1, \ldots$ of P. We will now exhibit a ν-thread in π. For every $i \in \mathbb{N}$ let $f(i)$ be the number of applications of rule $(\bigcirc)$ in π before Δ_i. We construct a linear time structure $\mathcal{K}$ as follows:

$$\forall i \in \mathbb{N} \text{ with } f(i) \neq f(i+1): \quad \mathcal{K}(f(i)) = \{\bar{l}_1, \ldots, \bar{l}_k\} \text{ iff } \Delta_i = \bigcirc\Gamma, l_1, \ldots, l_k$$

Consider for each Δ_i the formulas of Δ_i satisfied by $\mathcal{K}, f(i)$. Call them "true formulas". For each Δ_i there is at least one such true formula, because each Δ_i is a valid disjunction.

Every true formula is linked by the connection relation to a true formula in the preceding sequent; König's lemma thus delivers a thread comprising true formulas only. More formally, we obtain a sequence $\varphi_i \in \Delta_i$ such that $\varphi_0 = \varphi$ and $(\varphi_i, \varphi_{i+1}) \in \mathit{Con}(\Delta_i, \Delta_{i+1})$ and $\mathcal{K}, f(i) \models \varphi_i$. Assume that $(\varphi_i)_i$ is a μ-thread. There is an $i \in \mathbb{N}$ and a $\mu X.\psi \in \mathit{Sub}(\varphi)$ s.t. $\mathcal{K}, f(i) \models \mu X.\psi$. Furthermore, no greater $\nu Y.\psi'$ occurs on this thread after position i. According to Lemma 1 there is a $k \in \mathbb{N}$ s.t. $\mathcal{K}, f(i) \models \mu^k X.\psi$. Now note that the connection relation follows the definition of the approximants. Hence, by preservation of satisfaction along this thread, there must be a $i' > i$, s.t. $\mathcal{K}, f(i') \models \mu^0 X.\psi$ which is impossible. So, the thread $(\varphi_i)_i$ is a ν-thread as required. □

We remark without proof that the proof system is also complete for non-guarded formulas.

Let $\varphi_0 \in L_\mu^{\text{lin}}$ and $\nu X_1.\psi_1, \ldots, \nu X_n.\psi_n$ all ν-quantified formulas in $\mathit{FL}(\varphi_0)$, ordered s.t. $\nu X_i.\psi_i \in \mathit{Sub}(\psi_j)$ implies $i > j$. A *ν-signature* is a tuple $\zeta = (k_1, \ldots, k_n) \in (\mathbb{N} \cup \{\omega\})^n$. Note that the lexicographic ordering $<$ on ν-signatures is well-founded. We write $\zeta(i)$ for the i-th component of ζ, and $\mathcal{K}, i \models_\zeta \varphi$ if $\mathcal{K}, i$ is a model of the formula that results from φ when every $\nu X_i.\psi_i$ is interpreted by $\nu^{\zeta(i)} X_i.\psi_i$.

Theorem 2. *For all closed $\varphi \in L_\mu^{\text{lin}}$: if $\not\models \varphi$ then $\nvdash \varphi$.*

Proof. Suppose $\not\models \varphi$ but P is a proof for φ. Then there is a $\mathcal{K}$ s.t. $\mathcal{K}, 0 \not\models \varphi$. This can be used to construct a path $\pi = \Gamma_0, \Gamma_1, \ldots$ with inferences $r_0, r_1, \ldots$ in P, and a sequence $t_0 \leq t_1 \leq \ldots$ of positions in $\mathcal{K}$, s.t. $\mathcal{K}, t_i \not\models \Gamma_i$ (I), and whenever $(\alpha, \beta) \in \mathit{Con}_{r_i}(\Gamma_i, \Gamma_{i+1})$ and $\mathcal{K}, t_i \not\models_\zeta \alpha$ then $\mathcal{K}, t_{i+1} \not\models_\zeta \beta$ (II).

Let $\Gamma_0 := \varphi$ and $t_0 := 0$. If Γ_i and t_i have been constructed we regard the inference r_i leading to Γ_i (note that Γ_i cannot be an axiom). If $r_i = (\bigcirc)$ then $t_{i+1} := t_i + 1$. We put $t_{i+1} := t_i$ in all other cases. If $r_i \neq (\wedge)$ then Γ_i has a unique premiss $\Delta =: \Gamma_{i+1}$. In the case of $(\wedge)$ let $\psi_1 \wedge \psi_2 \in \Gamma_i$ be the principal formula of r_i. Let ζ be the least ν-signature s.t. $\mathcal{K}, t_i \not\models_\zeta \psi_1 \wedge \psi_2$ (it exists by Lemma 1 and (I)). Let Γ_{i+1} be the j-th premiss of r_i where $j \in \{1, 2\}$ s.t. $\mathcal{K}, t_i \not\models_\zeta \psi_j$. Clearly, this construction guarantees condition (II).

Since P is a proof, π must contain a ν-thread $(\varphi_i)_i$. For each $i \in \mathbb{N}$ let ζ_i be the minimal ν-signature s.t. $\mathcal{K}, t_i \not\models_{\zeta_i} \varphi_i$. Since $(\varphi_i, \varphi_{i+1}) \in \mathit{Con}_{r_i}(\Gamma_i, \Gamma_{i+1})$

we have $\zeta_{i+1} \leq \zeta_i$. Since there is an outermost fixpoint formula $\nu Z.\psi$ that gets unfolded infinitely often in this thread, there are infinitely many i s.t. $\zeta_i > \zeta_{i+1}$ which is a contradiction to the wellfoundedness of $<$. □

4 Deciding Validity I: Automata-Theoretic Method

We regard rule applications in a pre-proof for a L_μ^{lin} formula φ_0 as symbols of a finite alphabet. Formally, let $\Sigma_{\varphi_0} := \{\ \mathtt{L}(\varphi \wedge \psi), \mathtt{R}(\varphi \wedge \psi) \ \mid\ \varphi \wedge \psi \in \mathit{FL}(\varphi_0)\ \} \cup \{\mathtt{N}\} \cup \{\ \mathtt{P}(\varphi) \ \mid\ \varphi \in \mathit{FL}(\varphi_0)$ is of the form $\psi_1 \vee \psi_2, \sigma X.\psi$, or $\bigcirc\psi\ \}$.

An infinite branch $\pi = \Gamma_0, \Gamma_1, \ldots$ in a pre-proof for φ_0 induces a word $\pi' = r_0, r_1, \ldots \in \Sigma_{\varphi_0}^\omega$ in a straight-forward way:

$$r_i \ := \begin{cases} \mathtt{L}(\varphi \wedge \psi), & \text{if } \varphi \wedge \psi \text{ is principal in } \Gamma_i, \Gamma_{i+1} \text{ is left premiss of } \Gamma_i \\ \mathtt{R}(\varphi \wedge \psi), & \text{if } \varphi \wedge \psi \text{ is principal in } \Gamma_i, \Gamma_{i+1} \text{ is right premiss of } \Gamma_i \\ \mathtt{P}(\varphi), & \text{if } \varphi \text{ is principal in } \Gamma_i \text{ and not of the form } \bigcirc\varphi' \\ \mathtt{N}, & \text{if } (\Gamma_i, \Gamma_{i+1}) \text{ is an instance of } (\bigcirc) \end{cases}$$

We will not distinguish formally between a branch π and its induced ω-word π' over Σ_{φ_0}.

Next we define an NPA that accepts exactly those branches which contain a ν-thread. Let $\varphi_0 \in L_\mu^{\text{lin}}$, and define $\mathcal{A}_{\varphi_0} := (Q, \Sigma_{\varphi_0}, q_0, \delta, \Omega)$ where $Q := \mathit{FL}(\varphi_0)$ is the set of states with starting state $q_0 := \varphi_0$. The priority function $\Omega : Q \to \mathbb{N}$ is defined inductively as $\Omega(\psi_1 \vee \psi_2) = \Omega(\psi_1 \wedge \psi_2) := \max\{\Omega(\psi_1), \Omega(\psi_2)\}$; $\Omega(\bigcirc\varphi) := \Omega(\varphi)$; $\Omega(\sigma X.\varphi) := \Omega(\varphi)$ if $\Omega(\varphi)$ is odd and $\sigma = \mu$, or $\Omega(\varphi)$ is even and $\sigma = \nu$, and $\Omega(\sigma X.\varphi) := \Omega(\varphi) + 1$ otherwise; and $\Omega(\psi) := 0$ in all other cases. Here we assume that an NPA accepts a word if it has an accepting run in which the *greatest* priority occurring infinitely often is *even.*

Intuitively, $\mathcal{A}_{\varphi_0}$ traces a thread. The priority function ensures that the underlying word is accepted only if the guessed thread is a ν-thread. The transition relation therefore simply resembles the connection relation:

$$\begin{array}{llll}
\delta(\psi, r) & := \{\psi\} & \text{if } r \notin \{\mathtt{P}(\psi), \mathtt{L}(\psi), \mathtt{R}(\psi)\} & \\
\delta(\psi_1 \vee \psi_2, \mathtt{P}(\psi_1 \vee \psi_2)) & := \{\psi_1, \psi_2\} & \delta(\bigcirc\psi, \mathtt{N}) & := \{\psi\} \\
\delta(\psi_1 \wedge \psi_2, \mathtt{L}(\psi_1 \wedge \psi_2)) & := \{\psi_1\} & \delta(\bigcirc\psi, r) & := \{\bigcirc\psi\} \quad \text{if } r \neq \mathtt{N} \\
\delta(\psi_1 \wedge \psi_2, \mathtt{R}(\psi_1 \wedge \psi_2)) & := \{\psi_2\} & \delta(\sigma X.\varphi, \mathtt{P}(\sigma X.\varphi)) & := \{\varphi[\sigma X.\varphi/X]\}
\end{array}$$

Clearly, $|\mathcal{A}_{\varphi_0}|$, the number of states of $\mathcal{A}_{\varphi_0}$ is $|\varphi_0|$.

Lemma 2. *For all closed $\varphi_0 \in L_\mu^{\text{lin}}$ and all infinite branches π of a pre-proof for φ_0: $\pi \in L(\mathcal{A}_{\varphi_0})$ iff π contains a ν-thread.*

Proof. Let $\pi = \Gamma_0, \Gamma_1, \ldots$ be an infinite branch in a pre-proof for φ_0.

($\Leftarrow$) Suppose $\varphi_0, \varphi_1, \ldots$ is a ν-thread in π. Since δ is defined in accordance to the connection relation, this thread is also a run of $\mathcal{A}_{\varphi_0}$. By assumption, the outermost subformula of the form $\sigma X.\psi$ that occurs infinitely often in this thread is of type ν. Now note that the priority of an automaton state $\sigma X.\psi$ is even iff

$\sigma = \nu$, and outer formulas have greater priorities than inner ones. Hence, the greatest priority occurring infinitely often in this run is even, i.e. $\pi \in L(\mathcal{A}_{\varphi_0})$.

($\Rightarrow$) This is proved in the same way as the other direction. □

Furthermore, we define a (deterministic) Büchi automaton $\mathcal{B}_\varphi$ that accepts all the words which form a branch in a pre-proof for φ. In order to avoid notational clutter we simply assume that every branch in a pre-proof is infinite. Note that finite branches can be modeled by introducing a new final state in the automaton with a self-loop under any alphabet symbol.

Let $\mathcal{B}_\varphi := (2^{FL(\varphi)}, \Sigma_\varphi, \{\varphi\}, \delta, F)$ with $F := 2^{FL(\varphi)}$, and $\Delta \in \delta(\Gamma, r)$ iff Δ is a premiss of Γ in an application of rule r. The following is a direct consequence of the definition of a proof and Lemma 2.

Proposition 1. *For all $\varphi \in L_\mu^{\text{lin}}$: $\vdash \varphi$ iff $L(\mathcal{B}_\varphi) \subseteq L(\mathcal{A}_\varphi)$.*

This shows that validity in L_μ^{lin} can be decided using this proof system in an optimal way matching the known PSPACE lower bound [11].

Theorem 3. *Deciding whether or not $\vdash \varphi$ holds for a given $\varphi \in L_\mu^{\text{lin}}$ is in PSPACE.*

Proof. According to Proposition 1 it suffices to check the language $L(\mathcal{B}_\varphi) \cap \overline{L(\mathcal{A}_\varphi)}$ for non-emptiness. Let $n := |\varphi|$. Note that $|\mathcal{B}_\varphi| \leq 2^n$ and $|\mathcal{A}_\varphi| \leq n$. Using well-known automata-theoretic constructions and Savitch's Theorem this boils down to the emptiness test of an automaton $\mathcal{B} \times \overline{\mathcal{A}}$ which can be done in PSPACE. □

Proposition 1 yields a generic automata-theoretic method for deciding validity. We will compare various complementation and non-emptiness procedures for NBAs w.r.t. the incurring complexities. Note that every state of $\mathcal{B}_\varphi$ is final. Hence, the automaton $\mathcal{B}_\varphi \times \overline{\mathcal{A}_\varphi}$ can always be built in a simple product construction and is of the same type as $\overline{\mathcal{A}_\varphi}$.

construction	type of $\overline{\mathcal{A}_\varphi}$	$\lvert\mathcal{B}_\varphi \times \overline{\mathcal{A}_\varphi}\rvert$	emptiness test
Safra [10]	det. Streett	$2^{O(n^2 \log n)}$	$2^{O(n^2 \log n)}$
Sistla/Vardi/Wolper [12]	nondet. Büchi	$2^{O(n^4)}$	$2^{O(n^4)}$
Klarlund [5]	nondet. Büchi	$2^{O(n^2 \log n)}$	$2^{O(n^2 \log n)}$
Kupferman/Vardi [7]	weak alt. Büchi	$O(n^4)$	$2^{O(n^4)}$
Kupferman/Vardi [6]	weak alt. Büchi	$O(n^n)$	$2^{O(n^n)}$
Piterman [9]	det. parity	$2^{O(n^2 \log n)}$	$2^{O(n^2 \log n)}$

The index of the Streett or parity automaton is $O(n^2)$ in both cases. Note that the table lists the running times of a deterministic algorithm not using general theorems from complexity theory. We remark that using either of Safra's, Klarlund's or Piterman's construction improves asymptotically over Vardi's decision procedure for L_μ^{lin}. It also improves over the other $2^{O(n^2 \log n)}$ procedures mentioned in the introduction by being *a priori* deterministic. Furthermore, the procedures using Kupferman and Vardi's complementation can be implemented symbolically.

5 Deciding Validity II: Category-Theoretic Method

Let P be the (finite) set of priorities assigned to subformulas of φ_0 by the function Ω in Section 4. Let Γ and Δ be sequents. A morphism f from Γ to Δ written $f : \Gamma \to \Delta$ is a subset of $\Gamma \times \Delta \times P$. In this case, Γ is the domain of f and Δ is the codomain of f. If $f : \Gamma \to \Delta$ and $\Delta \to \Theta$ then the composition $f; g : \Gamma \to \Theta$ is the morphism defined by

$$f; g = \{(\gamma, \theta, p) \mid \exists\delta\ p_1\ p_2.(\gamma, \delta, p_1) \in f \wedge (\delta, \theta, p_2) \in g \wedge p = \max(p_1, p_2)\}$$

The identity morphism $\mathrm{id}_\Gamma : \Gamma \to \Gamma$ is given by $\mathrm{id}_\Gamma = \{(\gamma, \gamma, 0) \mid \gamma \in \Gamma\}$.

It is clear that composition is associative with identities as neutral elements and that therefore the sequents with morphisms form a category. If M is a set of morphisms we denote by $C(M)$ the set of morphisms obtained by closing M under composition and adding identities, i.e., the category generated by M. Notice that if M is a finite set so is $C(M)$ because there is only a finite number of sequents and morphisms.

A morphism $f : \Gamma \to \Delta$ is idempotent if $\Gamma = \Delta$ and $f; f = f$. An idempotent morphism f is *bad* if it does not contain a link of the form (φ, φ, p) with p even. A morphism should be viewed as a connection relation whose links are labeled with priorities.

Suppose that r is an instance of a rule occurring in a pre-proof of φ_0 with conclusion Γ and Δ one of its premisses. We define the morphism $f_{(\Gamma,r,\Delta)} : \Gamma \to \Delta$ by

$$f_{\Gamma,r,\Delta} = \{(\gamma, \delta, \Omega(\gamma)) \mid (\gamma, \delta) \in \mathit{Con}_r(\Gamma, \Delta)\}$$

If π is a finite branch occurring in a pre-proof then we obtain a morphism f_π by composing the morphisms $f_{\_,r,\_}$ that are associated with the sequents and rules occurring along π. If π begins at sequent Γ and ends at Δ then $(\gamma, \delta, p) \in f_\pi$ iff there is a run of $\mathcal{A}_{\varphi_0}$ on π beginning in state γ, ending in state δ and exhibiting p as the highest priority along this run.

Now let P be the generic pre-proof obtained as in the proof of Theorem 3. Note that in this pre-proof any sequent uniquely determines the proof rule which is (backwards-)applied to it.

Theorem 4. *Let M be the set of morphisms of the form $f_{\Gamma,r,\Delta}$ where r is a rule instance contained in the generic pre-proof P of φ_0. The following are equivalent.*
(a) φ_0 is valid.
(b) P is a proof.
(c) The closure $C(M)$ of M by composition contains no bad idempotent.

Proof. The equivalence between (a) and (b) is a direct consequence of Lemma 2 and Theorem 1. The interesting part is the equivalence between (b) and (c) and it is here that we draw inspiration from the graph-theoretic algorithm for size-change termination in [8] and in particular closely follow their proof idea.

(b)⇒(c) by contraposition: suppose that $C(M)$ contains a bad idempotent $f : \Delta \to \Delta$. Let π be the finite path in the generic proof P that led to f's being in $C(M)$. We use here the fact that every sequent uniquely determines its proof rule. Let ρ be a finite path in P leading from $\{\varphi_0\}$ to Δ and consider the infinite path $\rho;\pi;\pi;\pi;\ldots$ which is contained in P. We claim that this path is not accepted by $\mathcal{A}_{\varphi_0}$. Assume for a contradiction that there is an accepting run with n the highest (even) priority. Since Δ is finite there must exist $\delta \in \Delta$ such that the accepting run goes through δ and after consuming $\pi^i := \pi;\pi;\ldots;\pi$ (i times) for some $i > 0$ goes through δ again and moreover, the highest priority encountered along π^i is n. But this means that $(\delta,\delta,n) \in f$ contradicting the assumption that f was bad.

(c)⇒(b) Assume that $C(M)$ does not contain a bad idempotent. Let π be an infinite path in P. For $i < j$ let $\pi_{i,j}$ be the finite portion of π from i to j. By Ramsey's theorem there exists an infinite subset $U \subseteq \mathbb{N}$ and a morphism f such that $f_{\pi_{i,j}} = f$ whenever $i,j \in U$. It follows that f is idempotent. If f contains a link (δ,δ,n) with n even then we get a successful run on π with highest priority n simply by going through δ at each position in U and following the construction of $f_{\pi_{i,j}}$ in between. □

Theorem 4 directly leads to an algorithm for deciding validity of formulas: simply compute the set of morphisms M occurring in the generic pre-proof of φ_0, then iteratively calculate $C(M)$ and then look for a bad idempotent in $C(M)$. Of course, in practice one checks for bad idempotents already during the construction of M and $C(M)$ terminating the process immediately upon encountering one. The resulting algorithm is not in PSPACE since the size of $C(M)$ is exponential: $|C(M)| \leq 2^{n^2 \cdot p + 2n}$ where n is the size of the input formula and p is the highest priority of any subformula. Since $p \leq n$, and the runtime of our algorithm is quadratic in $|C(M)|$, it is also bounded by $2^{O(n^3)}$. We note that this also improves asymptotically on the runtime of Vardi's decision procedure for L_μ^{lin} [14].

6 Experimental Results

We have implemented two exponential time algorithms – the one based on an explicit computation of $C(M)$ and the one testing emptiness of a deterministic parity automaton using Piterman's determinisation procedure – in OCAML. In the following we present some experimental results obtained on three families of formulas.

$$\begin{aligned} \mathit{Include}_n \ &:= \ \nu X.\big(\underbrace{q \wedge \bigcirc(q \wedge \bigcirc(\ldots \bigcirc (q}_{2n \text{ times}} \wedge \bigcirc (\neg q \wedge \bigcirc X))\ldots)))\\ &\qquad\qquad\qquad \rightarrow \ \nu Z.\mu Y.(\neg q \wedge \bigcirc Z) \vee (q \wedge \bigcirc(q \wedge \bigcirc Y)) \end{aligned}$$

describes the valid statement $((aa)^n b)^\omega \subseteq ((aa)^* b)^\omega$, where the alphabet symbol a is the label $\{q\}$ and b is $\emptyset$. *Include*$_n$ is not LTL-definable for any $n \in \mathbb{N}$.

The next family is *Nester*$_n := \psi \vee \neg\psi$ where

$$\psi := \mu X_1.\nu X_2.\mu X_3.\ldots\sigma X_n.q_1 \vee \bigcirc\Big(X_1 \wedge \big(q_2 \vee \bigcirc(X_2 \wedge \ldots (q_n \vee \bigcirc X_n)\ldots)\big)\Big)$$

	Include$_n$		*Nester*$_n$		*Counter*$_n$	
n	$\lvert C(M)\rvert$	search	$\lvert C(M)\rvert$	search	$\lvert C(M)\rvert$	search
0	2,545	1,285	—	—	5	638
1	11,965	17,203	9	79	299	18,564
2	28,866	44,903	2,154	23,589	1,333	195,989
3	50,057	83,864	2,030,259	†	34,401	1,666,281
4	77,189	135,220	†		379,356	12,576,760
5	110,242	198,971			†	†

Fig. 2. Complexity measures for some example formulas

It is clearly valid and is chosen as an example with several alternating fixpoint constructs.

$$
\begin{aligned}
\mathit{Counter}_n \;:=\; (\bigvee_{i=0}^{n} \neg c_i) \;\vee\; & \Big(\mu X.\bigcirc X \;\vee\; (c_0 \not\leftrightarrow \bigcirc\neg c_0) \vee \\
& \bigvee_{i=1}^{n} \bigcirc c_i \;\not\leftrightarrow\; (c_i \wedge \neg c_{i-1}) \vee (c_{i-1} \wedge (\bigcirc c_{i-1} \leftrightarrow c_i))\Big)
\end{aligned}
$$

is not valid and is chosen because its smallest countermodel has size 2^{n+1}. Note that $\neg\mathit{Counter}_n$ formalises an $(n+1)$-bit counter.

Figure 2 presents empirical measures for the complexity of both procedures on the example formulas above. The columns labeled $|C(M)|$ contain the number of examined morphisms. Note that this is the number of all possible morphisms unless the input formula is not valid. The columns labeled "search" contain the number of search steps done in the emptiness test on the DPA $\mathcal{B} \times \overline{\mathcal{A}_\varphi}$. This is in general quadratic in the size of the automaton. The runtime was always around a few minutes but of course space is the limiting resource here. A dagger marks the tasks where our 1GB PCs ran out of memory.

7 Further Work

We would like to carry out a more systematic study of the practical usefulness of either algorithm. The examples from Section 6 were deliberately chosen to stress-test our approach. It may well be that formulas of the form $\varphi \Rightarrow \psi$ where φ describes an implementation of a system, e.g., a Mutex algorithm and where ψ is a specification of low quantifier nesting depth are feasible up to a much larger size. Should such experiments turn out promising one could then consider improving the treatment of the propositional part using BDDs or SAT-solvers, as well as using a symbolic implementation of the automata-theoretic algorithm. Notice namely that our decision procedures deal with propositional tautologies or consequences rather inefficiently basically by proof search in sequent calculus.

Also of interest could be optimisations using heuristics to guide the search for a countermodel as well as improvements on the theoretical side that reduce the size of the entire search space.

Furthermore, the proof system of Section 3 can be quite straightforwardly extended to capture validity of the modal μ-calculus: simply replace rule $(\bigcirc)$ with

$$\frac{\varphi, \Gamma}{[a]\varphi, \langle a\rangle\Gamma, \Delta}$$

This, however, introduces non-determinism (if a sequent contains several $[a]$-formulas), and countermodels become genuine trees. The automata-theoretic decision procedure of Section 4 can, in theory, easily be extended. Using Piterman's construction to determinise the automaton that recognises ν-threads, the product of this and the proof system becomes a parity game. Hence, validity in the modal μ-calculus can be solved using parity game solvers. On the other hand, whether or not the category-theoretical method of Section 5 can be extended to this framework is a nontrivial question.

References

1. H. Barringer, R. Kuiper, and A. Pnueli. A really abstract concurrent model and its temporal logic. In *Conf. Record of the 13th Annual ACM Symp. on Principles of Programming Languages, POPL'86*, pages 173–183. ACM, 1986.
2. J. C. Bradfield, J. Esparza, and A. Mader. An effective tableau system for the linear time μ-calculus. In *Proc. 23rd Int. Coll. on Automata, Languages and Programming, ICALP'96*, volume 1099 of *LNCS*, pages 98–109. Springer, 1996.
3. C. Dax. Games for the linear time μ-calculus. Master's thesis, Dep. of Computer Science, University of Munich, 2006. available from `http://www.tcs.ifi.lmu.de/lehre/da_fopra/Christian_Dax.pdf`.
4. R. Kaivola. A simple decision method for the linear time μ-calculus. In *Proc. Int. Workshop on Structures in Conc. Theory, STRICT'95*, pages 190–204, 1995.
5. N. Klarlund. Progress measures for complementation of ω-automata with applications to temporal logic. In *Proc. 32nd Annual Symp. on Foundations of Computer Science, FOCS'91*, pages 358–367. IEEE, 1991.
6. O. Kupferman and M. Y. Vardi. Weak alternating automata and tree automata emptiness. In *Proc. 30th Annual ACM Symp. on Theory of Computing, STOC'98*, pages 224–233. ACM Press, 1998.
7. O. Kupferman and M. Y. Vardi. Weak alternating automata are not that weak. *ACM Transactions on Computational Logic*, 2(3):408–429, 2001.
8. C. S. Lee, N. D. Jones, and A. M. Ben-Amram. The size-change principle for program termination. *j-SIGPLAN*, 36(3):81–92, 2001.
9. N.Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st Ann. IEEE Symp. on Logic in Computer Science, LICS'06*. IEEE Computer Society Press, 2006. To appear.
10. S. Safra. On the complexity of ω-automata. In *Proc. 29th Symp. on Foundations of Computer Science, FOCS'88*, pages 319–327. IEEE, 1988.
11. A. P. Sistla and E. M. Clarke. The complexity of propositional linear temporal logics. *Journal of the Association for Computing Machinery*, 32(3):733–749, 1985.
12. A. P. Sistla, M. Y. Vardi, and P. Wolper. The complementation problem for Büchi automata with applications to temporal logic. *TCS*, 49(2–3):217–237, 1987.
13. R. S. Streett and E. A. Emerson. An automata theoretic decision procedure for the propositional μ-calculus. *Information and Computation*, 81(3):249–264, 1989.
14. M. Y. Vardi. A temporal fixpoint calculus. In *Proc. Conf. on Principles of Programming Languages, POPL'88*, pages 250–259. ACM Press, 1988.

Tree Automata Make Ordinal Theory Easy

Thierry Cachat

LIAFA/CNRS UMR 7089 & Université Paris 7, France
txc@liafa.jussieu.fr

Abstract. We give a new simple proof of the decidability of the First Order Theory of $(\omega^{\omega^i}, +)$ and the Monadic Second Order Theory of $(\omega^i, <)$, improving the complexity in both cases. Our algorithm is based on tree automata and a new representation of (sets of) ordinals by (infinite) trees.

1 Introduction

The connections between automata and logic have been fruitful for many years, see [13] for an introduction. In 1960 Büchi [4] showed that sets of finite words can be equivalently defined by Monadic Second Order (MSO) formulas and by finite automata. This gives in particular a decision procedure for this logic. This result has been extended later to other classes of structures and automata: MSO over infinite words and Büchi automata in [5], MSO over transfinite ordinals and transfinite automata [6], MSO over the full binary tree and Rabin automata in [18], MSO over graphs of the Caucal hierarchy and graph automata [7,15].

The decidability of the first order logic over the integers with addition, also known as Presburger arithmetic, can be easily obtained by using finite automata reading binary representation of numbers. A central idea in all these results is that formulas can be represented by automata: by induction on the formula one can build an automaton accepting exactly the models of the formula. See [22] for a clear exposition of many of the previous results.

More recently many authors have used automata to improve the complexity of certain decisions procedures. In particular in [14] the Presburger arithmetic is considered and in [16] the first order theory of the ordinals with addition.

We address in this article the decision algorithms for the First Order theory (FO) of $(\omega^{\omega^i}, +)$ and the Monadic Second Order theory (MSO) of $(\omega^i, <)$ for any integer i. Our proposal is to use finite labeled trees to represent ordinals and infinite trees to represent sets of ordinals. Then one can use tree automata to represent formulas (namely, all their models). In this way we improve the best known complexity, and we hope that our constructions are easier to understand than previous ones. Note that already MSO$(\omega, +)$ is undecidable, and the decision procedure for MSO$(\omega, <)$ has a non elementary lower bound. In [12] trees are already used to represent ordinals, but only termination of preocesses is considered. Our infinite trees in Section 3 are close to those in [3], where only inclusion of languages is considered.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 285–296, 2006.

The paper is organized as follows. The next section is concerned with the first order theory. After recalling definitions we present our tree encoding and our decidability proof. In Section 3 the encoding is adapted to the Monadic Second Order theory, before comparisons to other results and techniques are given.

2 Decidability of the First Order Theory of $(\omega^\omega, +)$

2.1 Definitions: Ordinal Addition, First Order Logic, Tree Automata

We assume basic knowledge about ordinals, see e.g. [20,21]. An *ordinal* is a well and totally ordered set. It is either 0 or a successor ordinal of the form $\beta + 1$ or a limit ordinal. The first limit ordinal is denoted ω. For all ordinal α: $\beta < \alpha \Leftrightarrow \beta \in \alpha$ and $\alpha = \{\beta : \beta < a\}$. The set of natural numbers is identified with ω. Recall e.g. that $1 + \omega = \omega = 2\omega$ and $\omega + \omega^2 = \omega^2$ but $\omega + 1 \neq \omega \neq \omega 2$. By the Cantor Normal Form theorem, for all $0 < \alpha < \omega^\omega$ there exist unique integers $p, n_0, n_1, \ldots, n_p$ such that $n_p > 0$ and

$$\alpha = \omega^p n_p + \omega^{p-1} n_{p-1} + \cdots + \omega^1 n_1 + n_0 \ .$$

Ordinal addition has an *absorption* property: for any $p < p'$, $\omega^p + \omega^{p'} = \omega^{p'}$. Given two ordinals $\alpha = \omega^p n_p + \cdots + \omega^1 n_1 + n_0$ and $\alpha' = \omega^{p'} n'_{p'} + \cdots + \omega^1 n'_1 + n'_0$ both written in Cantor Normal Form, the ordinal $\alpha + \alpha'$ is

$$\omega^p n_p + \cdots + \omega^{p'} (n_{p'} + n'_{p'}) + \cdots + \omega^1 n'_1 + n'_0 \ .$$

Formulas of the *First Order Logic* (FO) over $(\omega^\omega, +)$ are built from

- a countable set of individual variables $x, y, z, \ldots$
- the addition $+$, seen as a ternary relation,
- the Boolean connectives $\neg$, $\wedge$, $\vee$, $\rightarrow$ and $\leftrightarrow$,
- first order quantification $\exists$ over individual variables ($\forall$ is seen as an abbreviation of $\neg\exists\neg$).

Example 1. The order relation $x \leq y$ can be easily defined as $\exists z : x + z = y$.
The relation $x < y$ is defined by $\neg(y \leq x)$.
The ordinal 0 is the only ordinal x such that $\neg\exists y : y < x$ or equivalently such that $x + x = x$.
The equality between x and y can be defined e.g. by $x \leq y \wedge y \leq x$.
The ordinal 1 is definable by $\phi(x) = (x > 0) \wedge \neg\exists y(0 < y \wedge y < x)$.

Example 2. The first limit ordinal, ω, is the only ordinal satisfying the formula

$$\begin{aligned}\varphi_1(x) = (x > 0) \wedge \forall y(y < x \rightarrow y + 1 < x) \wedge \\ \forall x'[(x' > 0) \wedge \forall y(y < x' \rightarrow y + 1 < x') \ \rightarrow \ x \leq x'] \ .\end{aligned}$$

Similarly and by induction ω^{i+1} is defined by

$$\begin{aligned}\varphi_{i+1}(x) = (x > 0) \wedge \forall y(y < x \rightarrow y + \omega^i < x) \wedge \\ \forall x'[(x' > 0) \wedge \forall y(y < x' \rightarrow y + \omega^i < x') \ \rightarrow \ x \leq x'] \ .\end{aligned}$$

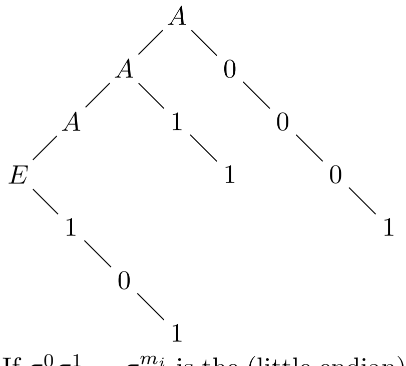
A *finite binary tree* T is a finite prefix closed subset of $\{a,b\}^*$. The root is the empty word ε, and for all $u \in \{a,b\}^*$, ua is the left successor of u and ub the right one. For simplicity we impose that each node has 0 or 2 successors: $\forall u \in \{a,b\}^*$, $ua \in T \Leftrightarrow ub \in T$. A leaf has no successor. Given a finite alphabet Σ, a Σ-labeled tree is a couple $\langle T, \lambda \rangle$ where T is a tree and λ is a function $\lambda : T \mapsto \Sigma$. A *tree automaton* is a tuple $(Q, \Sigma, \Delta, I, F)$ where Q is a finite set of states, Σ is a finite alphabet, $\Delta \subseteq Q \times \Sigma \times Q \times Q$ is the transition relation, $I \subseteq Q$ and $F \subseteq Q$ are the sets of initial and accepting states ("final states"). A Σ-labeled tree is accepted by such a tree automaton iff there exists a run $\rho : T \mapsto Q$ such that

$$\rho(\varepsilon) \in F, \text{ and } \forall u \in T: \begin{array}{cl} \text{either} & (\rho(u), \lambda(u), \rho(ua), \rho(ub)) \in \Delta \\ \text{or} & u \text{ is a leaf } (ua \notin T) \text{ and } \rho(u) \in I\ . \end{array}$$

This presentation is unusual: the labels at the leafs are not important in our constructions. These (bottom up) tree automata can be determinized by a usual subset construction. By exchanging initial and final states they can be seen as top down automata.

2.2 Binary Trees Representing Ordinals

Ordinals less than ω^ω can be easily represented by finite binary trees. The tree representing $\alpha = \omega^p n_p + \cdots + \omega^1 n_1 + n_0$ (where $n_p > 0$) has a leftmost branch of length (at least) p. At depth i on this branch a right branch is attached, holding the binary encoding of the number n_i. For example the ordinal $\omega^3.5 + \omega.3 + 8$ is represented essentially as the following tree.



The letter E marks the last position where there is a non zero right branch. We allow *all* possible ways to add dummy symbols # at the bottom of the tree. There are not represented on the picture, but they are needed for every node to have 0 or 2 successors (not 1). To be more formal the set of tree representations of a given ordinal $\alpha = \omega^p n_p + \cdots + \omega^1 n_1 + n_0$ is exactly the language accepted by the tree automaton to be defined next. The initial state is $q_\#$, the accepting state q_0.

If $\sigma_i^0 \sigma_i^1 \ldots \sigma_i^{m_i}$ is the (little endian) binary encoding of n_i: $n_i = \sum_{j=0}^{m_i} 2^j \sigma_i^j$, then the transitions are:

$$\begin{array}{rl} (q_i, A, q_{i+1}, p_i^0) \text{ if } i < p \text{ and } n_i > 0 & (p_i^j, \sigma_i^j, q_\#, p_i^{j+1}) \text{ if } j < m_i \\ (q_i, A, q_{i+1}, q_\#) \text{ if } i < p \text{ and } n_i = 0 & (p_i^j, \sigma_i^j, q_\#, q_\#) \text{ if } j = m_i \\ (q_i, E, q_\#, p_i^0) \text{ if } i = p & (q_\#, \#, q_\#, q_\#) \end{array}$$

In the special case where $\alpha = 0$ we have a transition $(q_0, \#, q_\#, q_\#)$. We denote T_α the tree coding an ordinal α.

2.3 Decidability Using Tree-Automata

We adapt a well known method for proving decidability of logic theories. A single tree over the alphabet $\{A, E, \#, 0, 1\}^k$ represents the values of k variables by superposing k corresponding trees (and adding dummy symbols $\#$). For every formula $\psi \in \mathrm{FO}(\omega^\omega, +)$ with free variables $x_1, \ldots, x_k$ we want to build a tree automaton over the alphabet $\{A, E, \#, 0, 1\}^k$ such that a tree is accepted by this automaton iff the corresponding valuation of the variables satisfies ψ. This can be done by induction on the formula. The case of Boolean connectives is easy using standard automata techniques of product and complementation, see [10]. Existential quantification results in projecting out the corresponding variable. The main point is to define an automaton recognizing the relation $x + y = z$, and this is easy with our coding.

In the following transitions $\begin{smallmatrix}\#\\ 1\\ 0\end{smallmatrix}$ represents a letter from $\{A, E, \#, 0, 1\}^3$ where the first component is $\#$, the second is 1 and the third is 0. These components are letters from T_x, T_y and T_z respectively. The symbols σ, δ represent digits from $\{0, 1\}$ and $*$ represents any letter. The accepting state is r. Because of the absorption property, above symbol E of T_y, trees T_y and T_z must coincide. State q_y checks that T_y and T_z coincide on the corresponding right branch. Similarly q_x checks that T_x and T_z coincide. State r_y checks that T_y and T_z coincide on the rest of the tree. Similarly r_x checks that T_x and T_z coincide.

$$(r, \begin{smallmatrix}\#\\ \#\\ \#\end{smallmatrix}, q_\#, q_\#) \qquad (q_\#, \begin{smallmatrix}\#\\ \#\\ \#\end{smallmatrix}, q_\#, q_\#)$$

$$(r, \begin{smallmatrix}A\\ A\\ A\end{smallmatrix}, r, q_y) \qquad (q_y, \begin{smallmatrix}*\\ \sigma\\ \sigma\end{smallmatrix}, q_\#, q_y) \qquad (q_y, \begin{smallmatrix}*\\ \#\\ \#\end{smallmatrix}, q_\#, q_y) \qquad (q_y, \begin{smallmatrix}\#\\ \#\\ \#\end{smallmatrix}, q_\#, q_\#)$$

$$(r, \begin{smallmatrix}E\\ A\\ A\end{smallmatrix}, r_y, q_y) \qquad (r_y, \begin{smallmatrix}\#\\ A\\ A\end{smallmatrix}, r_y, q_y) \qquad (r_y, \begin{smallmatrix}\#\\ E\\ E\end{smallmatrix}, q_\#, q_y)$$

$$(r, \begin{smallmatrix}A\\ E\\ A\end{smallmatrix}, r_x, q_0) \qquad (r_x, \begin{smallmatrix}A\\ \#\\ A\end{smallmatrix}, r_x, q_x) \qquad (r_x, \begin{smallmatrix}E\\ \#\\ E\end{smallmatrix}, q_\#, q_x)$$

$$(r, \begin{smallmatrix}E\\ E\\ E\end{smallmatrix}, q_\#, q_0) \qquad (q_x, \begin{smallmatrix}\sigma\\ *\\ \sigma\end{smallmatrix}, q_\#, q_x) \qquad (q_x, \begin{smallmatrix}\#\\ *\\ \#\end{smallmatrix}, q_\#, q_x) \qquad (q_x, \begin{smallmatrix}\#\\ \#\\ \#\end{smallmatrix}, q_\#, q_\#)$$

The states q_0 and q_1 are in charge of the binary addition with carries.

$$(q_0, \begin{smallmatrix}\sigma\\ \delta\\ \sigma \text{ XOR } \delta\end{smallmatrix}, q_\#, q_{\sigma \text{AND } \delta}) \qquad (q_0, \begin{smallmatrix}\sigma\\ \#\\ \sigma\end{smallmatrix}, q_\#, q_x) \qquad (q_0, \begin{smallmatrix}\#\\ \sigma\\ \sigma\end{smallmatrix}, q_\#, q_y)$$

$$(q_1, \begin{smallmatrix}\sigma\\ \delta\\ \neg(\sigma \text{ XOR } \delta)\end{smallmatrix}, q_\#, q_{\sigma \text{OR } \delta}) \qquad (q_1, \begin{smallmatrix}\sigma\\ \#\\ \neg\sigma\end{smallmatrix}, q_\#, q_\sigma) \qquad (q_1, \begin{smallmatrix}\#\\ \sigma\\ \neg\sigma\end{smallmatrix}, q_\#, q_\sigma)$$

$$(q_0, \begin{smallmatrix}\#\\ \#\\ \#\end{smallmatrix}, q_\#, q_\#) \qquad (q_1, \begin{smallmatrix}\#\\ \#\\ 1\end{smallmatrix}, q_\#, q_\#)$$

Some details are omitted here for the sake of simplicity. In state q_y, after reading $\#$ on the first component, one should check that only $\#$ appears. And the most significant bit of each number should be 1 to have a standard representation. It is left to the reader to add intermediate states to check that the trees T_x, T_y and T_z are well formed. That is needed when the automata defining T_x, T_y or T_z were obtained by complementation (see below). Let Tower stand for the "tower of exponentials" function, *i.e.*, $\text{Tower}(0, n) = n$ and $\text{Tower}(k+1, n) = 2^{\text{Tower}(k,n)}$.

Theorem 1. *The First Order Theory of* $(\omega^\omega, +)$ *is decidable in time* $\mathcal{O}(\mathit{Tower}(n, c))$*, for some constant* c*, where* n *is the length of the formula.*

To our knowledge the best known algorithm for deciding $\text{FO}(\omega^\omega, +)$ goes via a (linear) reduction to the Weak Monadic Second Order logic of $(\omega^\omega, <)$, which in turn is decidable in time $\mathcal{O}(\text{Tower}(6n, c'))$ [16]. See Section 3 for the definition of this logic.

Proof. By induction on the formula $\psi \in \text{FO}(\omega^\omega, +)$ one can construct a tree automaton $\mathcal{A}_\psi$ accepting exactly all valuations satisfying ψ. A valuation is here a tree labeled over $\{A, E, \#, 0, 1\}^k$, where k is the number of free variables in ψ.

- If ψ is an atomic proposition, it is of the form $x + y = z$ and we have seen how to construct $\mathcal{A}_\psi$.
- If ψ is of the form $\neg\psi'$, by induction $\mathcal{A}_{\psi'}$ is constructed. We can determinize and complement it [10], and intersect with the automaton describing the allowed representation of ordinals, to obtain $\mathcal{A}_\psi$.
- If ψ is of the form $\psi_1 \wedge \psi_2$, by induction $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$ are constructed. Rearrange the order of the variables, build the product of $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$. Declare a state $\langle q_1, q_2\rangle$ final iff both q_1 and q_2 are final. [10]
- Similarly if ψ is of the form $\psi_1 \vee \psi_2$, rearrange the variables, build the product and declare a state $\langle q_1, q_2\rangle$ final iff q_1 or q_2 is final.
- If ψ is of the form $\psi_1 \rightarrow \psi_2$, first determinize $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$, then build the product, and declare a state $\langle q_1, q_2\rangle$ final iff $(q_1 \in F_1) \Rightarrow (q_2 \in F_2)$.
- Similarly if ψ is of the form $\psi_1 \leftrightarrow \psi_2$, determinize $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$, build the product, and declare a state $\langle q_1, q_2\rangle$ final iff $(q_1 \in F_1) \Leftrightarrow (q_2 \in F_2)$.
- If ψ is of the form $\exists x \psi'$, then the input alphabet of the automaton $\mathcal{A}'_\psi$ is $\{A, E, \#, 0, 1\}^k$, where k is the number of free variables in ψ'. Project out the component corresponding to the variable x to get the automaton $\mathcal{A}_\psi$ that non-deterministically guesses the value of x.

At the end of the procedure it remains to determine whether $\mathcal{A}_\psi$ accepts a tree (labeled over an empty alphabet). This can be done in polynomial time by marking the states reachable from the initial states. Note that the cases of conjunction and disjunction does not need determinization. This is possible with a bottom up tree automaton, where the acceptance condition is checked only once, at the root.

Like for many automata based decision procedures, the most expensive step is the determinization of automata. It costs exponential time and the result is an automaton of exponential space. The number of steps of the construction is the number of Boolean connectives and quantifiers of the formula, whereas the constant c is essentially the number of states of the automaton for $x + y = z$.

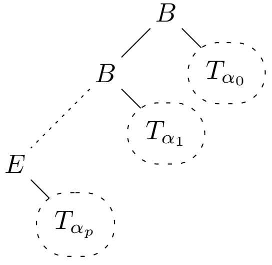
To slightly improve the complexity one can easily construct directly automata recognizing the relations $x = y$, $x < y$, $x \leq y$ of Example 1. Of course every ordinal ω^i can also be easily defined directly, without using the formulas of Example 2.

It is also possible to replace $\rightarrow$ and $\leftrightarrow$ by equivalent formulas using only $\neg$, $\wedge$ and $\vee$ and to push negations symbols inwards (using De Morgan's laws, etc). See [14] for a careful discussion about the cost of these transformations: they can increase the length of the formula and add new quantifiers. Here we do not assume that the formula is in prenex normal form.

2.4 Beyond ω^ω

By using a new letter (B) in the alphabet, it is possible to encode ordinals greater than ω^ω. Any ordinal $\beta < \omega^{\omega^2}$ can be uniquely written in the form

$$\omega^{\omega.p}\alpha_p + \cdots + \omega^{\omega.2}\alpha_2 + \omega^\omega \alpha_1 + \alpha_0 \ , \ \text{where } p < \omega, \alpha_i < \omega^\omega, \alpha_p > 0 \ .$$



and we can encode it as a tree where each T_{α_i} appears as a subtree. Namely the leftmost branch will have length p. At depth i on this branch the tree T_{α_i} is attached. The skeleton of the tree is depicted on the right. It is easy to see that a tree automaton can recognize the relation $x + y = z$, and that the proof of Theorem 1 carries over. Note that the letter B is used here only for clarity, one could use A instead.

This can be generalized by induction, and for all $i < \omega$ we can encode ordinals less than ω^{ω^i}.

Theorem 2. *For each $i < \omega$ there exists a constant c_i such that the First Order Theory of $(\omega^{\omega^i}, +)$ is decidable in time $\mathcal{O}(Tower(n, c_i))$, where n is the length of the formula.*

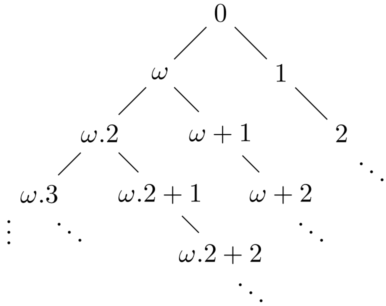
Note that the height of the tower of exponentials do not depend on i, and that c_i is linear in i. When considering FO$(\omega^{\omega^i}, +)$, even the ordinal 1 is coded by a tree of depth at least i: we need each tree to have the same skeleton to allow the automaton to proceed the addition locally. It was already noticed (without proof) in [11] that any ordinal $\alpha < \omega^{\omega^\omega}$ is tree-automatic, that is to say that the structure $(\alpha, <)$ —without addition— is definable using tree-automata. Moreover [11] proves that any tree-automatic ordinal is less than ω^{ω^ω}.

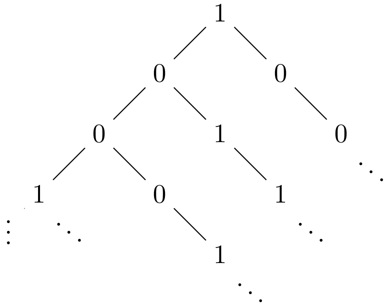
3 Monadic Second Order Theory of $(\omega^k, <)$

In this section we use full *infinite* binary trees. They are given by a mapping $\lambda : \{a, b\}^* \mapsto \Sigma$ for some finite alphabet Σ. Their domain is always $\{a, b\}^*$ so we do not need to mention it. One can adapt the idea of Section 2 to represent *sets* of ordinals. Given a subset $S \subseteq \omega^2$ it is represented by the tree $\lambda : \{a,b\}^* \mapsto \{0,1\}$ such that

$$\forall i,j \geq 0 : \lambda(a^i b^j) \in \{0,1\} \qquad \forall u \notin a^*b^* : \lambda(u) = \#$$
$$\forall i,j \geq 0 : \lambda(a^i b^j) = 1 \;\Leftrightarrow\; \omega.i + j \in S \ .$$

So positions are associated to ordinals according to the left tree of the next picture. Accordingly the right tree represents the set $\{0, \omega{+}1, \omega{+}2, \omega.2{+}2, \omega.3\}$. In this way one can represent any subset of ω^2.





Languages of infinite trees can be defined by top down *Muller automata* [17]. A Muller automaton $\mathcal{A}$ is a tuple $(Q, \Sigma, \Delta, I, \mathcal{F})$ where Q, Σ, Δ are the same as in Section 2, $I \subseteq Q$ is the set of initial states and $\mathcal{F} \subseteq \mathscr{P}(Q)$ is the acceptance component ($\mathscr{P}(Q)$ is the powerset of Q). A *run* of $\mathcal{A}$ on a Σ-labeled tree λ is a labeling $\rho : \{a,b\}^* \mapsto Q$ such that

$$\rho(\varepsilon) \in I \text{ and } \forall u \in T : (\rho(u), \lambda(u), \rho(ua), \rho(ub)) \in \Delta \ .$$

A run is accepting iff on each (infinite) branch of the run, the set of states appearing infinitely often is equal to one of the $F \in \mathcal{F}$. A tree is accepted iff there exists an accepting run. Muller automata cannot be determinized in general, but the class of languages accepted by Muller automata is closed under union, intersection, projection and complementation. In particular an automaton accepting all trees where only one node is labeled by 1 cannot be deterministic: it has to guess where is the 1.

Formulas of the *(full) Monadic Second Order Logic* (MSO) over $(\omega^\omega, <)$ are built from

- a countable set of first order variables $x, y, z, \ldots$
- a countable set of second order variables (in capitals) $X, Y, Z, \ldots$
- the order relation $(x < y)$ over first order variables,
- the membership relation $(x \in X)$, also written $X(x)$,
- the Boolean connectives $\neg$, $\wedge$ and $\vee$ ($\rightarrow$ and $\leftrightarrow$ are seen here as abbreviations),
- existential quantification ($\exists$) over first order *and* second order variables ($\forall$ is seen as an abbreviation of $\neg\exists\neg$).

The syntax of the *Weak Monadic Second Order Logic* (WMSO) is exactly the same, the difference is that second order variables are interpreted by *finite* subsets of the structure.

Example 3. The formulas of Example 1 above are also expressible in MSO$(\omega^\omega, <)$ because they do not need the addition. One can also define a relation $x = y + 1$. The next formula shows that the set of even ordinals (less than ω^ω) can be defined in MSO:

$$\exists X : \forall x \;\; (x \in X \leftrightarrow \neg(x + 1 \in X)) \wedge (\neg \exists y (x = y + 1) \rightarrow x \in X) \; .$$

We consider trees labeled over $\{0, 1\}^k$ where k is the number of first-order *and* second-order free variables. It should be clear that one can construct Muller automata recognizing the relations $x \in X$ and $x < y$. Note that for each first-order variable the automaton has to check that only one node in the tree is labeled by 1, *i.e.*, x is treated as a second-order variable $X = \{x\}$. See [2] for a clear exposition of a similar construction in the framework of ordinal automata.

Theorem 3. *The Monadic Second Order Theory of $(\omega^2, <)$ is decidable in time $\mathcal{O}(\mathit{Tower}(n, c))$, for some constant c, where n is the length of the formula.*

Recall that the upper bound of [16] is in $\mathcal{O}(\text{Tower}(6n, 1))$ for the *weak* variant WMSO$(\omega^\omega, <)$. Already MSO$(\omega, <)$ has a lower bound in $\Omega(\text{Tower}(n, d))$ for some constant $d > 0$ [19], so our bound is really tight.

Proof (sketch). We use again the well known method by induction on the structure of the formula $\psi \in$ MSO$(\omega^2, +)$.

- If ψ is an atomic proposition, it is clear how to construct $\mathcal{A}_\psi$.
- If ψ is of the form $\neg\psi'$, $\psi_1 \vee \psi_2$ or $\psi_1 \wedge \psi_2$, we use the fact that languages of Muller tree automata are closed under complementation, union and intersection.
- If ψ is of the form $\exists x \psi'$ or $\exists X \psi'$, we use the fact that languages of Muller tree automata are closed under projection.

The most expensive step is the complementation, it can be done in exponential time, and the result has also exponential size, see [17,22]. At the end the test of emptiness is also exponential.

Note that for the case of disjunction the automaton has to guess at the root which subformula can be true. For a formula $\psi = \psi_1 \rightarrow \psi_2$ we cannot do better than transform it into $\neg\psi_1 \vee \psi_2$. It is not correct to simply build the product of $\mathcal{A}_{\psi_1}$ and $\mathcal{A}_{\psi_2}$ and adapt the acceptance component, because the acceptance condition is checked independently on each branch.

Using an idea similar to that of Section 2.4, one can attach ω trees of the form presented above to a left-most branch to encode subsets of ω^3. This can be extended by induction to ω^i for all $i < \omega$.

Theorem 4. *For each $i < \omega$ there exists a constant c_i such that the Monadic Second Order Theory of $(\omega^i, <)$ is decidable in time $\mathcal{O}(\mathit{Tower}(n, c_i))$, where n is the length of the formula.*

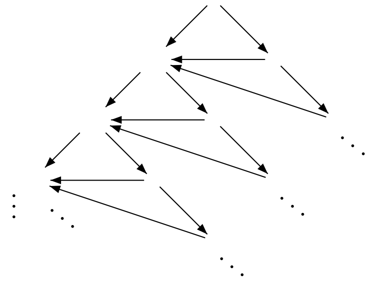
In other works such as [8,1] the emphasis is not placed on the complexity, but it seems that the complementation of ordinal automata is double exponential. It is open how to extend the tree encoding to subsets of ω^ω.

3.1 MSO-Interpretation. Comparison with Ordinal Automata

It is possible to put a different light on the previous constructions. The MSO theory of the full binary tree [22], called *S2S*, is build from the atomic propositions $S_a(x,y)$, $S_b(x,y)$ and $P(x)$, where S_a is the relation "left successor", S_b is "right successor" and P is a predicate that indicates that the label of a node is 1. In other words, given a labeled infinite tree $\lambda : \{a,b\}^* \mapsto \{0,1\}$ and $x, y \in \{a,b\}^*$:

$$S_a(x,y) \Leftrightarrow y = x.a\ , \qquad S_b(x,y) \Leftrightarrow y = x.b\ , \qquad P(x) \Leftrightarrow \lambda(x) = 1\ .$$

Recalling the left figure in page 291, the order among the ordinals/positions in the tree can be interpreted in S2S. That is, one can write a formula $\phi(x,y)$ such that $\phi(x,y)$ is true iff the ordinal of position x is less than that of y. It is easy if one first write formulas $\phi_a(x,y)$ and $\phi_b(x,y)$ that checks that y is a left descendant of x (resp. right descendant).



Alternatively one can see the ordering ω^2 as the transitive closure of the graph pictured on the right. Nevertheless concerning complexity it is better to construct dedicated automata as in the proof of Theorem 3. In other words the graphs of the orderings ω^i, $i < \omega$, are prefix-recognizable graphs [9]. It is open whether graphs of greater ordinals are in the Caucal hierarchy.

The usual proof that $\text{MSO}(\omega^\omega, <)$ is decidable uses ordinal automata reading ordinal words. An ordinal word of length α is a mapping $\alpha \mapsto \Sigma$, where Σ is a finite alphabet. An ordinal automaton has a state space Q, usual one-step transitions of the form $(q, \sigma, q') \in Q \times \Sigma \times Q$ and *limit transitions* of the form $(P, q') \in \mathscr{P}(Q) \times Q$, see e.g. [2]. They are a generalization of Muller (word) automata. A run is a mapping $\rho : \alpha + 1 \mapsto Q$. For a successor ordinal $\beta + 1$, $\rho(\beta + 1)$ is defined in the usual way. For a limit ordinal β, the state $\rho(\beta)$ is obtained by a limit transition according to the states appearing infinitely often "before" β.

We want to point out that a run of a Muller automaton on a tree representing $S \subseteq \omega^2$ is very similar to a run of length ω^2 of an ordinal automaton. Consider a node v at depth i on the left most branch. It corresponds to an ordinal $\omega.i$. The right-most branch from v must satisfy the Muller condition, and the state reached at the left successor of v is like the state reached at the limit transition at $\omega.(i+1)$. In this way we get a new proof that languages accepted by ordinal automata are closed under complementation, restricted to the case of words of length ω^j, for all $j < \omega$.

Comparing both approaches, we see that tree automata can not be determinized in general, they can be complemented, however, using an exponential construction. On the other side ordinal automata can be determinized (and complemented) using a doubly exponential construction, due to the nesting of Muller conditions. We are not aware of a better complementation algorithm for ordinal automata, see e.g. [8] for a more general result. The transformation from a tree automaton to an equivalent ordinal automaton according to our coding is very simple. The state space remains the same except for one extra final state for the last limit transition. If $(q, \lambda, q_a, q_b) \in \Delta$ in the tree automaton, add transitions (q, λ, q_b), and (P, q_a) for all $P \in \mathcal{F}$, where $\mathcal{F}$ is the Muller acceptance condition. The other way around is more complicated because the tree automaton has to guess what states are going to be visited infinitely often on the right branch, and then allow only these states to be visited infinitely often.

3.2 Weak MSO and FO

We introduce here new material to compare MSO and FO. Any ordinal β can be written in a unique way in the form

$$2^{\gamma_{n-1}} + \cdots + 2^{\gamma_0} \ , \quad \text{where} \quad (\gamma_{n-1}, \ldots, \gamma_0)$$

is a strictly decreasing sequence of ordinals. The set $\{\gamma_{n-1}, \ldots, \gamma_0\}$ is called the *2-development* of β. For example $2^\omega = \omega$, $2^{\omega.i+j} = 2^{\omega.i}.2^j = \omega^i.2^j$, $2^{\omega^2} = (2^\omega)^\omega = \omega^\omega$. Let E be the binary relation on ordinals such that $(x, y) \in E$ iff $x = 2^\gamma$ for some γ that belongs to the 2-development of y. It is known [6] that the theories WMSO$(\alpha, <)$ and FO$(2^\alpha, +, E)$ are equireducible in linear time. Recall that the (weak) theory WMSO is the monadic theory where only finite sets are considered. This mean that any formula of one of the logics can be translated into an equivalent formula of the other logic in linear time.

To extend Theorem 2 to the decidability of FO$(2^\alpha, +, E)$ for $\alpha = \omega^i$, we only need a tree automaton recognizing the relation E. The fact that $x = 2^\gamma$ is equivalent in our coding to the fact that exactly one label is 1 in the tree T_x, and $(x, y) \in E$ if moreover the same node is labeled by 1 in the tree T_y. The automaton recognizing E needs only three states, so the complexity bounds of Theorem 2 are not changed.

On the other side we have proved decidability of the *full* MSO theory of $(\omega^i, <)$ in Theorem 4. It remains to interpret WMSO in MSO. It is known in general how to construct a Muller tree automaton that checks that only finitely many nodes of a tree are labeled by 1. It is possible with only 2 states and can be used to adapt the proof of Theorem 3 to WMSO. Using this reduction, the complexity of the decision procedure of WMSO$(\omega^i, <)$ is in $\mathcal{O}(\text{Tower}(n+1, c'_i))$ for some (new) constant c'_i. Alternatively, using the property that every subset of an ordinal is also well ordered, it is possible to write an MSO formula that checks that a set of ordinals is finite. This formula should be used together with each second order quantification.

An extension of the previous tree-automata techniques to higher ordinals such as MSO$(\omega^\omega, <)$ would gives also tree-automata techniques for WMSO$(\omega^\omega, <)$ and then FO$(\omega^{\omega^\omega}, +, E)$, which is impossible [11] (see end of Section 2).

Related to the Cantor Normal Form (see Section 2), *any* ordinal β can yet be written in a unique way in the form

$$\alpha = \gamma.\omega^\omega + \omega^p n_p + \omega^{p-1} n_{p-1} + \cdots + \omega^1 n_1 + n_0 \ .$$

where $n_p > 0$. The ω-character of α is the sequence $(\sigma, n_p, \ldots, n_0)$ where $\sigma = 0$ if $\gamma = 0$, and $\sigma = 1$ if $\gamma > 0$. The theories $WMSO(\alpha, <)$ and $WMSO(\beta, <)$ are equal iff α and β have the same ω-character [6]. It follows that $\mathrm{FO}(2^\alpha, +, E)$ and $\mathrm{FO}(2^\beta, +, E)$ are equal iff α and β have the same ω-character.

4 Perspectives

We gave a new decision procedure for $\mathrm{FO}(\omega^{\omega^i}, +)$ and $\mathrm{MSO}(\omega^i, <)$ achieving better complexity bounds. We hope our constructions are easy to understand. As a byproduct we have a new proof of the complementation of ordinal automata restricted to words of length ω^i.

According to [11] (see end of Section 2) and Section 3.2 it is not possible to extend the tree-automata techniques to higher ordinals. But we would like to extend it to other linear orderings. A bi-infinite word is a mapping from the *relative* integers to a finite alphabet. It is easy to represent it as an infinite tree where only the right most and the left most branches are relevant. It seems easy to represent also orderings like $-\omega$ or $\omega \times (-\omega)$. Using a special letter, one could mark branches where the “reverse” ordering $-w$ is used. We conjecture that one can extend the results of Section 3 to more general linear orderings than just ordinals, and give a new proof of the results of [8].

Acknowledgments

Many thanks to Wolfgang Thomas for always saying that we have a proof without theorem (the proof that tree automata are closed under intersection, complementation and projection), to him and Stéphane Demri for pointing out some useful references and to the referees.

References

1. Nicolas Bedon. Finite automata and ordinals. *Theor. Comput. Sci.*, 156(1&2):119–144, 1996.
2. Nicolas Bedon. Logic over words on denumerable ordinals. *J. Comput. System Sci.*, 63(3):394–431, 2001.
3. Véronique Bruyère, Olivier Carton, and Géraud Sénizergues. Tree automata and automata on linear orderings. In *Proceedings of WORDS'03*, volume 27 of *TUCS Gen. Publ.*, pages 222–231. Turku Cent. Comput. Sci., Turku, 2003.
4. J. Richard Büchi. Weak second-order arithmetic and finite automata. *Zeitschrift für mathematische Logik und Grundlagen der Mathematik*, 6:66–92, 1960.
5. J. Richard Büchi. On a decision method in restricted second order arithmetic. In *Proceedings of the 1960 International Congress on Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.
6. J. Richard Büchi. Decision methods in the theory of ordinals. *Bull. Amer. Math. Soc.*, 71:767–770, 1965.

7. Thierry Cachat. Higher order pushdown automata, the Caucal hierarchy of graphs and parity games. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming, ICALP'03*, volume 2719 of *LNCS*, pages 556–569. Springer, 2003.
8. Olivier Carton and Chloe Rispal. Complementation of rational sets on scattered linear orderings of finite rank. In Martin Farach-Colton, editor, *LATIN*, volume 2976 of *LNCS*, pages 292–301. Springer, 2004.
9. Didier Caucal. On infinite terms having a decidable monadic theory. In *Proceedings of the 27th International Symposium on Mathematical Foundations of Computer Science 2002, MFCS 2002*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002.
10. Hubert Comon, Max Dauchet, Remi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available on: `http://www.grappa.univ-lille3.fr/tata`, 1997. release October, 1rst 2002.
11. Christian Delhommé. Automaticité des ordinaux et des graphes homogènes. *C. R. Math. Acad. Sci. Paris*, 339(1):5–10, 2004.
12. Nachum Dershowitz. Trees, ordinals and termination. In Marie-Claude Gaudel and Jean-Pierre Jouannaud, editors, *TAPSOFT*, volume 668 of *LNCS*, pages 243–250. Springer, 1993.
13. Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer, 2002.
14. Felix Klaedtke. On the automata size for Presburger arithmetic. In *Proceedings of the 19th Annual IEEE Symposium on Logic in Computer Science (LICS 2004)*, pages 110–119. IEEE Computer Society Press, 2004. A full version of the paper is available from the author's web page.
15. Orna Kupferman and Moshe Y. Vardi. An automata-theoretic approach to reasoning about infinite-state systems. In E. A. Emerson and A. P. Sistla, editors, *Proceedings of the 12th International Conference on Computer Aided Verification, CAV'00*, volume 1855 of *LNCS*, pages 36–52. Springer, 2000.
16. Françoise Maurin. Exact complexity bounds for ordinal addition. *Theoretical Computer Science*, 165(2):247–273, 1996.
17. Frank Nießner. Nondeterministic tree automata. In Grädel et al. [13], pages 135–152.
18. Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.
19. Klaus Reinhardt. The complexity of translating logic to finite automata. In Grädel et al. [13], pages 231–238.
20. Joseph G. Rosenstein. *Linear orderings*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], New York, 1982.
21. Wacław Sierpiński. *Cardinal and ordinal numbers*. Second revised edition. Monografie Matematyczne, Vol. 34. Pańtstowe Wydawnictwo Naukowe, Warsaw, 1965.
22. Mark Weyer. Decidability of S1S and S2S. In Grädel et al. [13], pages 207–230.

Context-Sensitive Dependency Pairs*

Beatriz Alarcón, Raúl Gutiérrez, and Salvador Lucas

DSIC, Universidad Politécnica de Valencia, Spain
{balarcon , rgutierrez , slucas}@dsic.upv.es

Abstract. Termination is one of the most interesting problems when dealing with context-sensitive rewrite systems. Although there is a good number of techniques for proving termination of context-sensitive rewriting (*CSR*), the dependency pair approach, one of the most powerful techniques for proving termination of rewriting, has not been investigated in connection with proofs of termination of *CSR*. In this paper, we show how to use dependency pairs in proofs of termination of *CSR*. The implementation and practical use of the developed techniques yield a novel and powerful framework which improves the current state-of-the-art of methods for proving termination of *CSR*.

Keywords: Dependency pairs, term rewriting, program analysis, termination.

1 Introduction

A *replacement map* is a mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ satisfying $\mu(f) \subseteq \{1, \ldots, k\}$, for each k-ary symbol f of a signature $\mathcal{F}$ [Luc98]. We use them to discriminate the argument positions on which the rewriting steps are allowed. In this way, for a given Term Rewriting System (TRS [Ohl02, Ter03]), we obtain a restriction of rewriting which we call *context-sensitive rewriting* (*CSR* [Luc98, Luc02]). In *CSR* we only rewrite μ-replacing subterms: t_i is a μ-replacing subterm of $f(t_1, \ldots, t_k)$ if $i \in \mu(f)$; every term t (as a whole) is μ-replacing by definition. With *CSR* we can *achieve* a terminating behavior with non-terminating TRSs, by pruning (all) infinite rewrite sequences. Proving termination of *CSR* has been recently recognized as an interesting problem with several applications in the fields of term rewriting and programming languages (see [DLMMU06, GM04, Luc02, Luc06]).

Several methods have been developed for proving termination of *CSR* under a replacement map μ for a given TRS $\mathcal{R}$ (i.e., for proving the μ*-termination* of $\mathcal{R}$). In particular, a number of transformations which permit to treat termination of *CSR* as a standard termination problem have been described (see

* This work has been partially supported by the EU (FEDER) and the Spanish MEC, under grant TIN 2004-7943-C04-02, the Generalitat Valenciana under grant GV06/285, and the ICT for EU-India Cross-Cultural Dissemination ALA/95/23/2003/077-054 project. Beatriz Alarcón was partially supported by the Spanish MEC under FPU grant AP2005-3399.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 297–308, 2006.

[GM04, Luc06] for recent surveys). Direct techniques like polynomial orderings and the context-sensitive version of the recursive path ordering have also been investigated [BLR02, GL02, Luc04b, Luc05]. Up to now, however, the *dependency pairs method* [AG00, GAO02, GTS04, HM04], one of the most powerful techniques for proving termination of rewriting, has not been investigated in connection with proofs of termination of *CSR*. In this paper, we address this problem.

Roughly speaking, given a TRS $\mathcal{R}$, the dependency pairs associated to $\mathcal{R}$ conform a new TRS $\mathsf{DP}(\mathcal{R})$ which (together with $\mathcal{R}$) determines the so-called *dependency chains* whose finiteness or infiniteness characterize termination of $\mathcal{R}$. Given a rewrite rule $l \to r$, we get dependency pairs $l^\sharp \to s^\sharp$ for all subterms s of r which are rooted by a defined symbol[1]; the notation $t^\sharp$ for a given term t means that the root symbol f of t is *marked* thus becoming $f^\sharp$ (often just capitalized: F). A chain of dependency pairs is a sequence $u_i \to v_i$ of dependency pairs such that $\sigma(v_i)$ rewrites to $\sigma(u_{i+1})$ for some substitution σ and $i \geq 1$. The dependency pairs can be presented as a *dependency graph*, where the absence of infinite chains can be analyzed by considering the *cycles* in the graph. These basic intuitions are valid for *CSR*, although some important differences arise.

Example 1. Consider the following TRS $\mathcal{R}$ [GM99, Example 1]:

```
c -> a      f(a,b,X) -> f(X,X,X)
c -> b
```

together with $\mu(\mathtt{f}) = \{3\}$. As shown by Giesl and Middeldorp, among all existing transformations for proving termination of *CSR*, only the *complete* Giesl and Middeldorp's transformation [GM04] (yielding a TRS $\mathcal{R}^\mu_C$) could be used in this case, but no concrete proof of termination for $\mathcal{R}^\mu_C$ is known yet. Furthermore, $\mathcal{R}^\mu_C$ has 13 dependency pairs and the dependency graph contains many cycles. In contrast, $\mathcal{R}$ has only *one* context-sensitive (CS-)dependency pair

```
F(a,b,X) -> F(X,X,X)
```

and the corresponding dependency graph has *no* cycle (due to the replacement restrictions, since we extend μ by $\mu(\mathtt{F}) = \{3\}$). As we show below, a direct (and automatic) proof of μ-termination of $\mathcal{R}$ is easy now.

Basically, the subterms in the right-hand sides of the rules which are considered to build the CS-dependency pairs must be *μ-replacing* terms. However, this is not sufficient to obtain a correct approximation. The following example shows the need of a new kind of dependency pairs.

Example 2. Consider the following TRS $\mathcal{R}$:

```
a -> c(f(a))
f(c(X)) -> X
```

together with $\mu(\mathtt{c}) = \varnothing$ and $\mu(\mathtt{f}) = \{1\}$. There is no μ-replacing subterm s in the right-hand sides of the rules which is rooted by a defined symbol. Thus, there is no ‘regular’ dependency pair. We could wrongly conclude that $\mathcal{R}$ is μ-terminating, which is not true:

[1] A symbol f is said to be *defined* in a TRS $\mathcal{R}$ if $\mathcal{R}$ contains a rule $f(l_1, \ldots, l_k) \to r$.

$$\mathtt{f(\underline{a})} \hookrightarrow_\mu \underline{\mathtt{f(c(f(a)))}}\ \mathtt{f(\underline{a})} \hookrightarrow_\mu \cdots$$

Indeed, we must add the following dependency pair

```
F(c(X)) -> X
```

which would not be allowed in Arts and Giesl's approach [AG00] because the right-hand side is a variable.

After some preliminaries in Section 2, Section 3 introduces the general framework to compute and use context-sensitive dependency pairs for proving termination of *CSR*. The introduction of a new kind of dependency pairs (as in Example 2) leads to a new notion of context-sensitive dependency *chain*. We prove the correctness and completeness of the new approach, i.e., our dependency pairs approach fully characterize termination of *CSR*. We also show how to use term orderings for proving termination of *CSR* by means of the new approach. Furthermore, we are properly extending Arts and Giesl's approach: whenever $\mu(f) = \{1, \ldots, k\}$ for all k-ary symbols $f \in \mathcal{F}$, *CSR* and ordinary rewriting coincide; coherently, our results boil down into the standard results for the dependency pair approach. Section 4 shows how to compute the (estimated) context-sensitive dependency graph and investigates how to use term orderings together with the dependency graph to achieve automatic proofs of termination of *CSR* within the dependency pairs approach. Section 5 adapts Hirokawa and Middeldorp's subterm criterion [HM04] to *CSR*. Section 6 concludes.

2 Preliminaries

Throughout the paper, $\mathcal{X}$ denotes a countable set of variables and $\mathcal{F}$ denotes a signature, i.e., a set of function symbols $\{\mathtt{f}, \mathtt{g}, \ldots\}$, each having a fixed arity given by a mapping $ar : \mathcal{F} \to \mathbb{N}$. The set of terms built from $\mathcal{F}$ and $\mathcal{X}$ is $\mathcal{T}(\mathcal{F}, \mathcal{X})$. Positions $p, q, \ldots$ are represented by chains of positive natural numbers used to address subterms of t. Given positions p, q, we denote their concatenation as $p.q$. If p is a position, and Q is a set of positions, $p.Q = \{p.q \mid q \in Q\}$. We denote the topmost position by Λ. The set of positions of a term t is $\mathcal{P}os(t)$. Positions of non-variable symbols in t are denoted as $\mathcal{P}os_{\mathcal{F}}(t)$, and $\mathcal{P}os_{\mathcal{X}}(t)$ are the positions of variables. The subterm at position p of t is denoted as $t|_p$ and $t[s]_p$ is the term t with the subterm at position p replaced by s. We write $t \trianglerighteq s$ if $s = t|_p$ for some $p \in \mathcal{P}os(t)$ and $t \triangleright s$ if $t \trianglerighteq s$ and $t \neq s$. The symbol labelling the root of t is denoted as $root(t)$. A *context* is a term $C \in \mathcal{T}(\mathcal{F} \cup \{\Box\}, \mathcal{X})$ with zero or more 'holes' $\Box$ (a fresh constant symbol).

A rewrite rule is an ordered pair (l, r), written $l \to r$, with $l, r \in \mathcal{T}(\mathcal{F}, \mathcal{X})$, $l \notin \mathcal{X}$ and $\mathcal{V}ar(r) \subseteq \mathcal{V}ar(l)$. The left-hand side (*lhs*) of the rule is l and r is the right-hand side (*rhs*). A TRS is a pair $\mathcal{R} = (\mathcal{F}, R)$ where R is a set of rewrite rules. Given $\mathcal{R} = (\mathcal{F}, R)$, we consider $\mathcal{F}$ as the disjoint union $\mathcal{F} = \mathcal{C} \uplus \mathcal{D}$ of symbols $c \in \mathcal{C}$, called *constructors* and symbols $f \in \mathcal{D}$, called *defined functions*, where $\mathcal{D} = \{root(l) \mid l \to r \in R\}$ and $\mathcal{C} = \mathcal{F} - \mathcal{D}$.

Context-sensitive rewriting. A mapping $\mu : \mathcal{F} \to \mathcal{P}(\mathbb{N})$ is a *replacement map* (or $\mathcal{F}$-map) if $\forall f \in \mathcal{F},\ \mu(f) \subseteq \{1, \ldots, ar(f)\}$ [Luc98]. Let $M_{\mathcal{F}}$ be the set of all

$\mathcal{F}$-maps (or $M_{\mathcal{R}}$ for the $\mathcal{F}$-maps of a TRS $(\mathcal{F}, R)$). A binary relation R on terms is μ-monotonic if $t\,R\,s$ implies $f(t_1,\ldots,t_{i-1},t,\ldots,t_k)\,R\,f(t_1,\ldots,t_{i-1},s,\ldots,t_k)$ for all $f \in \mathcal{F}$, $i \in \mu(f)$, and $t,s,t_1,\ldots,t_k \in \mathcal{T}(\mathcal{F},\mathcal{X})$. The set of *$\mu$-replacing positions* $\mathcal{P}os^{\mu}(t)$ of $t \in \mathcal{T}(\mathcal{F},\mathcal{X})$ is: $\mathcal{P}os^{\mu}(t) = \{\Lambda\}$, if $t \in \mathcal{X}$ and $\mathcal{P}os^{\mu}(t) = \{\Lambda\} \cup \bigcup_{i\in\mu(root(t))} i.\mathcal{P}os^{\mu}(t|_i)$, if $t \notin \mathcal{X}$. The set of *replacing* variables of t is $\mathcal{V}ar^{\mu}(t) = \{x \in \mathcal{V}ar(t) \mid \exists p \in \mathcal{P}os^{\mu}(t), t|_p = x\}$. The μ-replacing subterm relation $\trianglerighteq_\mu$ is given by $t \trianglerighteq_\mu s$ if there is $p \in \mathcal{P}os^{\mu}(t)$ such that $s = t|_p$. We write $t \rhd_\mu s$ if $t \trianglerighteq_\mu s$ and $t \neq s$. In *context-sensitive rewriting* (*CSR* [Luc98]), we (only) contract *replacing* redexes: t μ-rewrites to s, written $t \hookrightarrow_\mu s$ (or $t \hookrightarrow_{\mathcal{R},\mu} s$ and even $t \hookrightarrow s$), if $t \xrightarrow{p}_{\mathcal{R}} s$ and $p \in \mathcal{P}os^{\mu}(t)$. A TRS $\mathcal{R}$ is μ-terminating if $\hookrightarrow_\mu$ is terminating. A term t is μ-terminating if there is no infinite μ-rewrite sequence $t = t_1 \hookrightarrow_\mu t_2 \hookrightarrow_\mu \cdots \hookrightarrow_\mu t_n \hookrightarrow_\mu \cdots$ starting from t. A pair $(\mathcal{R},\mu)$ where $\mathcal{R}$ is a TRS and $\mu \in M_{\mathcal{R}}$ is often called a CS-TRS.

Dependency pairs. Given a TRS $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ a new TRS $\mathsf{DP}(\mathcal{R}) = (\mathcal{F}^\sharp, D(R))$ of *dependency pairs* for $\mathcal{R}$ is given as follows: if $f(t_1,\ldots,t_m) \to r \in R$ and $r = C[g(s_1,\ldots,s_n)]$ for some defined symbol $g \in \mathcal{D}$ and $s_1,\ldots,s_n \in \mathcal{T}(\mathcal{F},\mathcal{X})$, then $f^\sharp(t_1,\ldots,t_m) \to g^\sharp(s_1,\ldots,s_n) \in D(R)$, where $f^\sharp$ and $g^\sharp$ are new fresh symbols (called *tuple* symbols) associated to defined symbols f and g respectively [AG00]. Let $\mathcal{D}^\sharp$ be the set of tuple symbols associated to symbols in $\mathcal{D}$ and $\mathcal{F}^\sharp = \mathcal{F} \cup \mathcal{D}^\sharp$. As usual, for $t = f(t_1,\ldots,t_k) \in \mathcal{T}(\mathcal{F},\mathcal{X})$, we write $t^\sharp$ to denote the *marked* term $f^\sharp(t_1,\ldots,t_k)$. Conversely, given a marked term $t = f^\sharp(t_1,\ldots,t_k)$, where $t_1,\ldots,t_k \in \mathcal{T}(\mathcal{F},\mathcal{X})$, we write $t^\natural$ to denote the term $f(t_1,\ldots,t_k) \in \mathcal{T}(\mathcal{F},\mathcal{X})$. Given $T \subseteq \mathcal{T}(\mathcal{F},\mathcal{X})$, let $T^\sharp$ be the set $\{t^\sharp \mid t \in T\}$.

A reduction pair $(\succsim, \sqsupset)$ consists of a stable and weakly monotonic quasi-ordering $\succsim$, and a stable and well-founded ordering $\sqsupset$ satisfying either $\succsim \circ \sqsupset \subseteq \sqsupset$ or $\sqsupset \circ \succsim \subseteq \sqsupset$. Note that *monotonicity is not required* for $\sqsupset$.

3 Context-Sensitive Dependency Pairs

Let $\mathcal{M}_{\infty,\mu}$ be a set of minimal non-μ-terminating terms in the following sense: t belongs to $\mathcal{M}_{\infty,\mu}$ if t is non-μ-terminating and every strict *μ-replacing* subterm s of t (i.e., $t \rhd_\mu s$) is μ-terminating. Obviously, if $t \in \mathcal{M}_{\infty,\mu}$, then $root(t)$ is a defined symbol. The following proposition establishes that, given a minimal non-μ-terminating term $t \in \mathcal{M}_{\infty,\mu}$, there are two ways for an infinite μ-rewrite sequence to proceed. The first one is by using 'visible' parts of the rules which correspond to μ-replacing subterms in the right-hand sides which are rooted by a defined symbol. The second one is by showing up 'hidden' non-μ-terminating subterms which are activated by *migrating* variables in a rule $l \to r$, i.e., variables $x \in \mathcal{V}ar^{\mu}(r) - \mathcal{V}ar^{\mu}(l)$ which are *not* μ-replacing in the left-hand side l but become μ-replacing in the right-hand side r.

Proposition 1. *Let $\mathcal{R} = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and $\mu \in M_{\mathcal{R}}$. Then for all $t \in \mathcal{M}_{\infty,\mu}$, there exist $l \to r \in R$, a substitution σ and a term $u \in \mathcal{M}_{\infty,\mu}$ such* *that* $t \overset{>\Lambda}{\hookrightarrow^*} \sigma(l) \overset{\Lambda}{\to} \sigma(r) \trianglerighteq_\mu u$ *and either (1) there is a μ-replacing subterm s of r such that $u = \sigma(s)$, or (2) there is $x \in \mathcal{V}ar^\mu(r) - \mathcal{V}ar^\mu(l)$ such that $\sigma(x) \trianglerighteq_\mu u$.*

Proposition 1 motivates the following.

Definition 1. *Let $\mathcal{R} = (\mathcal{F}, R) = (\mathcal{C} \uplus \mathcal{D}, R)$ be a TRS and $\mu \in M_\mathcal{R}$. We define $\mathsf{DP}(\mathcal{R},\mu) = \mathsf{DP}_\mathcal{F}(\mathcal{R},\mu) \cup \mathsf{DP}_\mathcal{X}(\mathcal{R},\mu)$ to be the set of* context-sensitive *dependency pairs (CS-DPs) where:*

$$\mathsf{DP}_\mathcal{F}(\mathcal{R},\mu) = \{l^\sharp \to s^\sharp \mid l \to r \in R, r \trianglerighteq_\mu s, root(s) \in \mathcal{D}, l \ntriangleright_\mu s\}$$

and $\mathsf{DP}_\mathcal{X}(\mathcal{R},\mu) = \{l^\sharp \to x \mid l \to r \in R, x \in \mathcal{V}ar^\mu(r) - \mathcal{V}ar^\mu(l)\}$. We extend $\mu \in M_\mathcal{F}$ into $\mu^\sharp \in M_{\mathcal{F}^\sharp}$ by $\mu^\sharp(f) = \mu(f)$ if $f \in \mathcal{F}$, and $\mu^\sharp(f^\sharp) = \mu(f)$ if $f \in \mathcal{D}$.

A rule $l \to r$ of a TRS $\mathcal{R}$ is μ-conservative if $\mathcal{V}ar^\mu(r) \subseteq \mathcal{V}ar^\mu(l)$, i.e., it does not contain migrating variables; $\mathcal{R}$ is μ-conservative if all its rules are (see [Luc06]). The following result is immediate from Definition 1.

Proposition 2. *If $\mathcal{R}$ is a μ-conservative TRS, then $\mathsf{DP}(\mathcal{R},\mu) = \mathsf{DP}_\mathcal{F}(\mathcal{R},\mu)$.*

Therefore, in order to deal with μ-conservative TRSs $\mathcal{R}$ we only need to consider the ‘classical’ dependency pairs in $\mathsf{DP}_\mathcal{F}(\mathcal{R},\mu)$.

Example 3. Consider the TRS $\mathcal{R}$:

```
g(X) -> h(X)      h(d) -> g(c)      c -> d
```

together with $\mu(\mathtt{g}) = \mu(\mathtt{h}) = \varnothing$ [Zan97, Example 1]. $\mathsf{DP}(\mathcal{R},\mu)$ is:

```
G(X) -> H(X)      H(d) -> G(c)
```

with $\mu^\sharp(\mathtt{G}) = \mu^\sharp(\mathtt{H}) = \varnothing$.

If the TRS $\mathcal{R}$ contains non-μ-conservative rules, then we also need to consider dependency pairs with variables in the right-hand side.

Example 4. Consider the TRS $\mathcal{R}$ [Zan97, Example 5]:

```
if(true,X,Y) -> X        f(X) -> if(X,c,f(true))
if(false,X,Y) -> Y
```

with $\mu(\mathtt{if}) = \{1,2\}$. Then, $\mathsf{DP}(\mathcal{R},\mu)$ is:

```
F(X) -> IF(X,c,f(true))        IF(false,X,Y) -> Y
```

with $\mu^\sharp(\mathtt{F}) = \{1\}$ and $\mu(\mathtt{IF}) = \{1,2\}$.

Now we introduce the notion of chain of CS-DPs.

Definition 2 (Chain of CS-DPs). *Let $(\mathcal{R},\mu)$ be a CS-TRS. Given $\mathcal{P} \subseteq \mathsf{DP}(\mathcal{R},\mu)$, an $(\mathcal{R},\mathcal{P},\mu^\sharp)$-chain is a finite or infinite sequence of pairs $u_i \to v_i \in \mathcal{P}$, for $i \geq 1$ such that there is a substitution σ satisfying both:*

1. *$\sigma(v_i) \hookrightarrow^*_{\mathcal{R},\mu^\sharp} \sigma(u_{i+1})$, if $u_i \to v_i \in \mathsf{DP}_\mathcal{F}(\mathcal{R},\mu)$, and*
2. *if $u_i \to v_i = u_i \to x_i \in \mathsf{DP}_\mathcal{X}(\mathcal{R},\mu)$, then there is $s_i \in \mathcal{T}(\mathcal{F},\mathcal{X})$ such that $\sigma(x_i) \trianglerighteq_\mu s_i$ and $s_i^\sharp \hookrightarrow^*_{\mathcal{R},\mu^\sharp} \sigma(u_{i+1})$.*

for $i \geq 1$. Here, as usual we assume that different occurrences of dependency pairs do not share any variable (renamings are used if necessary).

An $(\mathcal{R}, \mathcal{P}, \mu^\sharp)$-chain with $u_1 \to v_1 \in \mathcal{P}$ as heading dependency pair is called minimal *if $\sigma(u_1)^\natural \in \mathcal{M}_{\infty,\mu}$ and all dependency pairs in $\mathcal{P}$ occur infinitely often.*

Remark 1. When an $(\mathcal{R}, \mathsf{DP}(\mathcal{R}, \mu), \mu^\sharp)$-chain is written for a given substitution σ, we write $\sigma(u) \hookrightarrow_{\mathsf{DP}(\mathcal{R},\mu),\mu^\sharp} \sigma(v)$ for steps which use a dependency pair $u \to v \in \mathsf{DP}_{\mathcal{F}}(\mathcal{R}, \mu)$ but we rather write $\sigma(u) \hookrightarrow_{\mathsf{DP}(\mathcal{R},\mu),\mu^\sharp} s^\sharp$ for steps which use a dependency pair $u \to x \in \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$, where s is as in Definition 2.

In the following, we use $\mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu)$ to denote the subset of dependency pairs in $\mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$ whose migrating variables occur on non-μ-replacing immediate subterms in the left-hand side:

$$\mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu) = \{f^\sharp(u_1, \ldots, u_k) \to x \in \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu) \mid \exists i, 1 \leq i \leq k, i \notin \mu(f^\sharp), x \in \mathcal{V}ar(u_i)\}$$

For instance, $\mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu) = \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$ for the CS-TRS $(\mathcal{R}, \mu)$ in Example 4. For this subset of CS-dependency pairs, we have the following.

Proposition 3. *There is no infinite $(\mathcal{R}, \mathcal{P}, \mu^\sharp)$-chain with $\mathcal{P} \subseteq \mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu)$.*

The following result establishes the correctness of the context-sensitive depenency pairs approach.

Theorem 1 (Correctness). *Let $\mathcal{R}$ be a TRS and $\mu \in M_{\mathcal{R}}$. If there is no infinite $(\mathcal{R}, \mathsf{DP}(\mathcal{R}, \mu), \mu^\sharp)$-chain, then $\mathcal{R}$ is μ-terminating.*

As an immediate consequence of Theorem 1 and Proposition 3, we have the following.

Corollary 1. *Let $\mathcal{R}$ be a TRS and $\mu \in M_{\mathcal{R}}$. If $\mathsf{DP}(\mathcal{R}, \mu) = \mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu)$, then $\mathcal{R}$ is μ-terminating.*

Example 5. Consider the following TRS $\mathcal{R}$ [Luc98, Example 15]

```
and(true,X) -> X          first(0,X) -> nil
and(false,Y) -> false     first(s(X),cons(Y,Z)) -> cons(Y,first(X,Z))
if(true,X,Y) -> X         from(X) -> cons(X,from(s(X)))
if(false,X,Y) -> Y
add(0,X) -> X
add(s(X),Y) -> s(add(X,Y))
```

with $\mu(\texttt{cons}) = \mu(\texttt{s}) = \mu(\texttt{from}) = \varnothing$, $\mu(\texttt{add}) = \mu(\texttt{and}) = \mu(\texttt{if}) = \{1\}$, and $\mu(first) = \{1, 2\}$. Then, $\mathsf{DP}(\mathcal{R}, \mu) = \mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu)$ is:

```
ADD(0,X) -> X          IF(true,X,Y) -> X
AND(true,X) -> X       IF(false,X,Y) -> Y
```

Thus, by Corollary 1 we conclude the μ-termination of $\mathcal{R}$.

Now we prove that the previous CS-dependency pairs approach is not only correct but also complete for proving termination of *CSR*.

Theorem 2 (Completeness). *Let $\mathcal{R}$ be a TRS and $\mu \in M_{\mathcal{R}}$. If $\mathcal{R}$ is μ-terminating, then there is no infinite $(\mathcal{R}, \mathsf{DP}(\mathcal{R}, \mu), \mu^\sharp)$-chain.*

Corollary 2 (Characterization of μ-termination). *Let $\mathcal{R}$ be a TRS and $\mu \in M_{\mathcal{R}}$. $\mathcal{R}$ is μ-terminating iff there is no infinite $(\mathcal{R}, \mathsf{DP}(\mathcal{R}, \mu), \mu^\sharp)$-chain.*

In the dependency pairs approach, the absence of infinite chains is checked by finding a *reduction pair* $(\succeq, \sqsupset)$ which is compatible with the rules and the dependency pairs [AG00]. In our setting, we can relax the monotonicity requirements and use *μ-reduction pairs* $(\gtrsim, \sqsupset)$ where $\gtrsim$ is a stable and μ-monotonic quasi-ordering which is compatible with the well-founded and stable ordering $\sqsupset$, i.e., $\gtrsim \circ \sqsupset \subseteq \sqsupset$ or $\sqsupset \circ \gtrsim \subseteq \sqsupset$. The following result shows how to use μ-reduction pairs for proving μ-termination. This is the context-sensitive counterpart of [AG00, Theorem 7]; however, a number of remarkable differences arise due to the treatment of the dependency pairs in $\mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$. Basically, we need to ensure that the quasi-ordering is able to 'look' for a μ-replacing subterm inside the instantiation $\sigma(x)$ of a migrating variable x (hence we require $\trianglerighteq_\mu \subseteq \gtrsim$) and also connect a term which is rooted by defined symbol f and the corresponding dependency pair which is rooted by $f^\sharp$ (hence the requirement $f(x_1, \ldots, x_k) \gtrsim f^\sharp(x_1, \ldots, x_k)$).

Theorem 3. *Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS, $\mu \in M_{\mathcal{F}}$. Then, $\mathcal{R}$ is μ-terminating if and only if there is a μ-reduction pair $(\gtrsim, \sqsupset)$ such that,*

1. *$l \gtrsim r$ for all $l \to r \in R$,*
2. *$u \sqsupset v$ for all $u \to v \in \mathsf{DP}_{\mathcal{F}}(\mathcal{R}, \mu)$, and*
3. *whenever $\mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu) \neq \varnothing$ we have that $\trianglerighteq_\mu \subseteq \gtrsim$, where $\trianglerighteq_\mu$ is the μ-replacing subterm relation on $\mathcal{T}(\mathcal{F}, \mathcal{X})$, and*
   (a) $u \; (\gtrsim \cup \sqsupset) \; v$ for all $u \to v \in \mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu)$, $u \sqsupset v$ for all $u \to v \in \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu) - \mathsf{DP}^1_{\mathcal{X}}(\mathcal{R}, \mu)$, and $f(x_1, \ldots, x_k) \gtrsim f^\sharp(x_1, \ldots, x_k)$ for all $f \in \mathcal{D}$, or
   (b) $u (\gtrsim \cup \sqsupset) v$ for all $u \to v \in \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$ and $f(x_1, \ldots, x_k) \sqsupset f^\sharp(x_1, \ldots, x_k)$ for all $f \in \mathcal{D}$.

4 Context-Sensitive Dependency Graph

As noticed by Arts and Giesl, the analysis of infinite sequences of dependency pairs can be made by looking at (the cycles $\mathfrak{C}$ of) the *dependency graph* associated to the TRS $\mathcal{R}$. The nodes of the dependency graph are the dependency pairs in $\mathsf{DP}(\mathcal{R})$; there is an arc from a dependency pair $u \to v$ to a dependency pair $u' \to v'$ if there are substitutions σ and θ such that $\sigma(v) \to^*_{\mathcal{R}} \theta(u')$.

Similarly, in the *context-sensitive (CS-)dependency graph*:

1. There is an arc from a dependency pair $u \to v \in \mathsf{DP}_{\mathcal{F}}(\mathcal{R}, \mu)$ to a dependency pair $u' \to v' \in \mathsf{DP}(\mathcal{R}, \mu)$ if there are substitutions σ and θ such that $\sigma(v) \hookrightarrow^*_{\mathcal{R}, \mu^\sharp} \theta(u')$.
2. There is an arc from a dependency pair $u \to v \in \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$ to *each* dependency pair $u' \to v' \in \mathsf{DP}(\mathcal{R}, \mu)$.

Note that the use of $\mu^\sharp$ (which restricts reductions on the arguments of the dependency pair symbols $f^\sharp$) is essential: given a set of dependency pairs associated to a CS-TRS $(\mathcal{R},\mu)$, we have less arcs between them due to the presence of such replacement restrictions.

Example 6. Consider the CS-TRS in Example 1. $\mathsf{DP}(\mathcal{R},\mu)$ is:

```
    F(a,b,X) -> F(X,X,X)
```

with $\mu^\sharp(\mathtt{F}) = \{3\}$. Although the dependency graph contains a cycle (due to $\sigma(\mathtt{F(X,X,X)}) \to^* \sigma(\mathtt{F(a,b,Y)})$ for $\sigma(X) = \sigma(Y) = \mathtt{c}$), the CS-dependency graph contains *no* cycle because it is *not* possible to $\mu^\sharp$-reduce $\theta(\mathtt{F(X,X,X)})$ into $\theta(\mathtt{F(a,b,Y)})$ for any substitution θ (due to $\mu^\sharp(\mathtt{F}) = \{3\}$).

As noticed by Arts and Giesl, the presence of an infinite chain of dependency pairs correspond to a cycle in the dependency graph (but not vice-versa).

Again, as an immediate consequence of Theorem 1 and Proposition 3, we have the following.

Corollary 3. *Let $\mathcal{R}$ be a TRS, $\mu \in M_{\mathcal{R}}$ and $\mathfrak{C} \subseteq \mathsf{DP}^1_{\mathcal{X}}(\mathcal{R},\mu)$ be a cycle. Then, there is no minimal $(\mathcal{R},\mathfrak{C},\mu^\sharp)$-chain.*

According to this, and continuing Example 6, we conclude the μ-termination of $\mathcal{R}$ in Example 1.

4.1 Estimating the CS-Dependency Graph

In general, the (context-sensitive) dependency graph of a TRS is *not* computable and we need to use some approximation of it. Following [AG00], we describe how to approximate the CS-dependency graph of a CS-TRS $(\mathcal{R},\mu)$. Let $\textsc{Cap}^\mu$ be given as follows: let D be a set of defined symbols (in our context, $D = \mathcal{D} \cup \mathcal{D}^\sharp$):

$$\textsc{Cap}^\mu(x) = x \quad \text{if } x \text{ is a variable}$$
$$\textsc{Cap}^\mu(f(t_1,\ldots,t_k)) = \begin{cases} y & \text{if } f \in D \\ f([t_1]^f_1,\ldots,[t_k]^f_1) & \text{otherwise} \end{cases}$$

where y is intended to be a new, fresh variable which has not yet been used and given a term s, $[s]^f_i = \textsc{Cap}^\mu(s)$ if $i \in \mu(f)$ and $[s]^f_i = s$ if $i \notin \mu(f)$. Let $\textsc{Ren}^\mu$ given by: $\textsc{Ren}^\mu(x) = y$ if x is a variable and $\textsc{Ren}^\mu(f(t_1,\ldots,t_k)) = f([t_1]^f_1,\ldots,[t_k]^f_k)$ for evey k-ary symbol f, where given a term $s \in \mathcal{T}^\sharp(\mathcal{F},\mathcal{X})$, $[s]^f_i = \textsc{Ren}^\mu(s)$ if $i \in \mu(f)$ and $[s]^f_i = s$ if $i \notin \mu(f)$. Then, we have an arc from $u_i \to v_i$ to $u_j \to v_j$ if $\textsc{Ren}^\mu(\textsc{Cap}^\mu(v_i))$ and u_j unify; following [AG00], we say that v_i and u_j are *μ-connectable.* The following result whose proof is similar to that of [AG00, Theorem 21] (we only need to take into account the replacement restrictions indicated by the replacement map μ) formalizes the correctness of this approach.

Proposition 4. *Let $(\mathcal{R},\mu)$ be a CS-TRS. If there is an arc from $u \to v$ to $u' \to v'$ in the CS-dependency graph, then v and u' are μ-connectable.*

Example 7. (Continuing Ex. 6) Since $\textsc{Ren}^{\mu^\sharp}(\textsc{Cap}^{\mu^\sharp}(\texttt{F(X,X,X)})) = \texttt{F(X,X,Z)}$ and `F(a,b,Y)` do not unify, we conclude (and this can be easily implemented) that the CS-dependency graph for the CS-TRS $(\mathcal{R}, \mu)$ in Example 1 has no cycle.

4.2 Checking μ-Termination with the Dependency Graph

For the cycles in the dependency graph, the absence of infinite chains is checked by finding (possibly different) *reduction pairs* $(\succeq_{\mathfrak{C}}, \sqsupset_{\mathfrak{C}})$ for each cycle $\mathfrak{C}$ [GAO02, Theorem 3.5]. In our setting, we use μ-reduction pairs.

Theorem 4 (Use of the CS-dependency graph). *Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS, $\mu \in M_{\mathcal{F}}$. Then, $\mathcal{R}$ is μ-terminating if and only if for each cycle $\mathfrak{C}$ in the context-sensitive dependency graph there is a μ-reduction pair $(\gtrsim_{\mathfrak{C}}, \sqsupset_{\mathfrak{C}})$ such that, $R \subseteq \gtrsim_{\mathfrak{C}}$, $\mathfrak{C} \subseteq \gtrsim_{\mathfrak{C}} \cup \sqsupset_{\mathfrak{C}}$, and*

1. *If $\mathfrak{C} \cap \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu) = \varnothing$, then $\mathfrak{C} \cap \sqsupset_{\mathfrak{C}} \neq \varnothing$*
2. *If $\mathfrak{C} \cap \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu) \neq \varnothing$, then $\trianglerighteq_\mu \subseteq \gtrsim_{\mathfrak{C}}$ (where $\trianglerighteq_\mu$ is the μ-replacing subterm relation on $\mathcal{T}(\mathcal{F}, \mathcal{X})$), and*
   (a) $\mathfrak{C} \cap \sqsupset_{\mathfrak{C}} \neq \varnothing$ and $f(x_1, \ldots, x_k) \gtrsim_{\mathfrak{C}} f^\sharp(x_1, \ldots, x_k)$ for all $f^\sharp$ in $\mathfrak{C}$, or
   (b) $f(x_1, \ldots, x_k) \sqsupset_{\mathfrak{C}} f^\sharp(x_1, \ldots, x_k)$ for all $f^\sharp$ in $\mathfrak{C}$.

Following Hirokawa and Middeldorp, the practical use of Theorem 4 concerns the so-called *strongly connected components*(SCCs) of the dependency graph, rather than the cycles themselves (which are exponentially many) [HM04, HM05]. A strongly connected component in the (CS-)dependency graph is a *maximal cycle*, i.e., it is not contained in any other cycle. According to Hirokawa and Middeldorp, when considering an SCC $\mathfrak{C}$, we *remove* from $\mathfrak{C}$ those pairs $u \to v$ satisfying $u \sqsupset v$. Then, we recompute the SCCs with the remaining pairs in the CS-dependency graph and start again (see [HM05, Section 4]). In our setting, it is not difficult to see that, if the condition $f(x_1, \ldots, x_k) \sqsupset_{\mathfrak{C}} f^\sharp(x_1, \ldots, x_k)$ for all $f \in \mathcal{D}$ holds for a given cycle $\mathfrak{C}$, then we can remove from $\mathfrak{C}$ *all* dependency pairs in $\mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$, thus continuing from $\mathfrak{C} - \mathsf{DP}_{\mathcal{X}}(\mathcal{R}, \mu)$.

Example 8. Consider the CS-TRS $(\mathcal{R}, \mu)$ in Example 4 and $\mathsf{DP}(\mathcal{R}, \mu)$:

```
F(X) -> IF(X,c,f(true))
IF(false,X,Y) -> Y
```

with $\mu^\sharp(\texttt{F}) = \{1\}$ and $\mu^\sharp(\texttt{IF}) = \{1, 2\}$. These two CS-dependency pairs form the only cycle in the CS-dependency graph. The μ-reduction pair $(\geq, >)$ induced by the polynomial interpretation

$$\begin{array}{rrr} [\texttt{c}] = [\texttt{true}] = 0 & [\texttt{f}](x) = x & [\texttt{F}](x) = x \\ [\texttt{false}] = 1 & [\texttt{if}](x, y, z) = x + y + z & [\texttt{IF}](x, y, z) = x + z \end{array}$$

can be used to prove the μ-termination of $\mathcal{R}$.

The use of *argument filterings*, which is standard in the current formulations of the dependency pairs method, also adapts without changes to the context-sensitive setting. This is a simple consequence of [AG00, Theorem 11] (using μ-monotonicity instead of monotonicity for the quasi-orderings is not a problem).

5 Subterm Criterion

In [HM04], Hirokawa and Middeldorp introduce a very interesting *subterm criterion* which permits to ignore certain cycles of the dependency graph.

Definition 3. [HM04] *Let $\mathcal{R}$ be a TRS and $\mathfrak{C} \subseteq \mathsf{DP}(\mathcal{R})$ such that every dependency pair symbol in $\mathfrak{C}$ has positive arity. A* simple projection *for $\mathfrak{C}$ is a mapping π that assigns to every k-ary dependency pair symbol $f^\sharp$ in $\mathfrak{C}$ an argument position $i \in \{1, \ldots, k\}$. The mapping that assigns to every term $f^\sharp(t_1, \ldots, t_k) \in \mathcal{T}^\sharp(\mathcal{F}, \mathcal{X})$ with $f^\sharp$ a dependency pair symbol in $\mathcal{R}$ its argument position $\pi(f^\sharp)$ is also denoted by π.*

In the following result, for a simple projection π and $\mathfrak{C} \subseteq \mathsf{DP}(\mathcal{R}, \mu)$, we let $\pi(\mathfrak{C}) = \{\pi(u) \to \pi(v) \mid u \to v \in \mathfrak{C}\}$. Note that $u, v \in \mathcal{T}^\sharp(\mathcal{F}, \mathcal{X})$, but $\pi(u), \pi(v) \in \mathcal{T}(\mathcal{F}, \mathcal{X})$.

Theorem 5. *Let $\mathcal{R}$ be a TRS and $\mu \in M_\mathcal{R}$. Let $\mathfrak{C} \subseteq \mathsf{DP}_\mathcal{F}(\mathcal{R}, \mu)$ be a cycle. If there exists a simple projection π for $\mathfrak{C}$ such that $\pi(\mathfrak{C}) \subseteq \trianglerighteq_\mu$, and $\pi(\mathfrak{C}) \cap \rhd_\mu \neq \varnothing$, then there is no minimal $(\mathcal{R}, \mathfrak{C}, \mu^\sharp)$-chain.*

Note that the result is restricted to cycles which do *not* include dependency pairs in $\mathsf{DP}_\mathcal{X}(\mathcal{R}, \mu)$. The following result provides a kind of generalization of the subterm criterion to simple projections which only consider *non-μ-replacing* arguments of tuple symbols.

Theorem 6. *Let $\mathcal{R} = (\mathcal{F}, R)$ be a TRS, $\mu \in M_\mathcal{F}$ and $\mathfrak{C} \subseteq \mathsf{DP}_\mathcal{F}(\mathcal{R}, \mu)$ be a cycle. Let $\gtrsim$ be a stable quasi-ordering on terms whose strict and stable part $>$ is well-founded and π be a simple projection for $\mathfrak{C}$ such that for all $f^\sharp$ in $\mathfrak{C}$, $\pi(f^\sharp) \notin \mu^\sharp(f^\sharp)$ and $\pi(\mathfrak{C}) \subseteq \gtrsim$.*

1. *If $\mathfrak{C} \cap \mathsf{DP}_\mathcal{X}(\mathcal{R}, \mu) = \varnothing$ and $\mathfrak{C} \cap > \neq \varnothing$, then there is no minimal $(\mathcal{R}, \mathfrak{C}, \mu^\sharp)$-chain.*
2. *If $\mathfrak{C} \cap \mathsf{DP}_\mathcal{X}(\mathcal{R}, \mu) \neq \varnothing$, $\trianglerighteq_\mu \subseteq \gtrsim$ (where $\trianglerighteq_\mu$ is the μ-replacing subterm relation on $\mathcal{T}(\mathcal{F}, \mathcal{X})$), and*
   (a) $\mathfrak{C} \cap > \neq \varnothing$ and $f(x_1, \ldots, x_k) \gtrsim x_{\pi(f^\sharp)}$ for all $f \in \mathcal{D}$ such that $f^\sharp$ is in $\mathfrak{C}$, or
   (b) $f(x_1, \ldots, x_k) > x_{\pi(f^\sharp)}$ for all $f \in \mathcal{D}$ such that $f^\sharp$ is in $\mathfrak{C}$,
   then there is no minimal $(\mathcal{R}, \mathfrak{C}, \mu^\sharp)$-chain.

Example 9. Consider the CS-TRS $(\mathcal{R}, \mu)$ in Example 3. $\mathsf{DP}(\mathcal{R}, \mu)$ is:

```
G(X) -> H(X)
H(d) -> G(c)
```

where $\mu^\sharp(\mathtt{G}) = \mu^\sharp(\mathtt{H}) = \varnothing$. The dependency graph contains a single cycle including both of them. The only simple projection is $\pi(\mathtt{G}) = \pi(\mathtt{H}) = 1$. Since $\pi(\mathtt{G(X)}) = \pi(\mathtt{H(X)})$, we only need to guarantee that $\pi(\mathtt{H(d)}) = \mathtt{d} > \mathtt{c} = \pi(\mathtt{G(c)})$ holds for a stable and well-founded ordering $>$. This is easily fulfilled by, e.g., a polynomial ordering.

6 Conclusions

We have shown how to use dependency pairs in proofs of termination of *CSR*. The implementation and practical use of the developed techniques yield a novel and powerful framework which improves the current state-of-the-art of methods for proving termination of *CSR*. Some interesting differences arise which can be summarized as follows: in sharp contrast to the standard dependency pairs approach, where all dependency pairs have tuple symbols $f^\sharp$ both in the left- and right-hand sides, we have dependency pairs having a single *variable* in the right-hand side. These variables reflect the effect of the *migrating* variables into the termination behavior of *CSR*. This leads to a new definition of chain of context-sensitive dependency pairs which also differs from the standard approach in that we have to especially deal with such migrating variables. As in Arts and Giesl's approach, the presence or absence of infinite chains of dependency pairs from $\mathsf{DP}(\mathcal{R},\mu)$ characterizes the μ-terminaton of $\mathcal{R}$ (Theorems 1 and 2). Furthermore, we are also able to use term orderings to ensure the absence of infinite chains of context-sensitive dependency pairs (Theorem 3). In fact, we are properly extending Arts and Giesl's approach: whenever $\mu(f) = \{1,\ldots,k\}$ for all k-ary symbols $f \in \mathcal{F}$, *CSR* and ordinary rewriting coincide and all these results and techniques boil down into well-known results and techniques for the dependency pairs approach.

Regarding the practical use of the CS-dependency pairs in proofs of termination of *CSR*, we have shown how to build and use the corresponding CS-dependency graph to either prove that the rules of the TRS and the cycles in the CS-dependency graph are compatible with some reduction pair (Theorem 4) or to prove that there are cycles which do not need to be considered at all (Theorems 5 and 6). We have implemented these ideas as part of the termination tool MU-TERM [AGIL07, Luc04a]. We refer the reader to [AGIL07] for details about the practical impact of the techniques developed in this paper. From this preliminary results, we can well conclude that the CS-dependency pairs can play in *CSR* the (practical and theoretical) role than dependency pairs play in rewriting.

There are many other aspects of the dependency pairs approach which are also worth to be considered and eventually extended to *CSR* (e.g., narrowing refinements, modularity issues, innermost computations, usable rules, ...). These aspects provide an interesting subject for future work.

References

[AG00] T. Arts and J. Giesl. Termination of Term Rewriting Using Dependency Pairs *Theoretical Computer Science*, 236:133-178, 2000.

[AGIL07] B. Alarcón, R. Gutiérrez, J. Iborra, and S. Lucas. Proving Termination of Context-Sensitive Rewriting with MU-TERM. *Electronic Notes in Theoretical Computer Science*, *to appear*, 2007.

[BLR02] C. Borralleras, S. Lucas, and A. Rubio. Recursive Path Orderings can be Context-Sensitive. In *Proc. of CADE'02*, LNAI 2392:314-331, Springer-Verlag, Berlin, 2002.

[DLMMU06] F. Durán, S. Lucas, J. Meseguer, C. Marché, and X. Urbain. Proving Operational Termination of Membership Equational Programs. *Higher-Order and Symbolic Computation*, to appear, 2006.

[GAO02] J. Giesl, T. Arts, and E. Ohlebusch Modular Termination Proofs for Rewriting Using Dependency Pairs. *Journal of Symbolic Computation* 34(1):21-58, 2002.

[GL02] B. Gramlich and S. Lucas. Simple termination of context-sensitive rewriting. In *Proc. of RULE'02*, pages 29-41, ACM Press, New York, 2002.

[GM99] J. Giesl and A. Middeldorp. Transforming Context-Sensitive Rewrite Systems. In *Proc. of RTA'99*, LNCS 1631:271-285, Springer-Verlag, Berlin, 1999.

[GM04] J. Giesl and A. Middeldorp. Transformation techniques for context-sensitive rewrite systems. *Journal of Functional Programming*, 14(4): 379-427, 2004.

[GTS04] J. Giesl, R. Thiemann, and P. Schneider-Kamp. The Dependency Pair Framework: Combining Techniques for Automated Termination Proofs. In *Proc. of LPAR'04*, LNCS 3452:301-331, Springer-Verlag, Berlin, 2004.

[HM04] N. Hirokawa and A. Middeldorp. Dependency Pairs Revisited. In *Proc. of RTA'04*, LNCS 3091:249-268, Springer-Verlag, Berlin, 2004.

[HM05] N. Hirokawa and A. Middeldorp. Automating the dependency pair method. *Information and Computation*, 199:172-199, 2005.

[Luc98] S. Lucas. Context-sensitive computations in functional and functional logic programs. *Journal of Functional and Logic Programming*, 1998(1):1-61, January 1998.

[Luc02] S. Lucas. Context-sensitive rewriting strategies. *Information and Computation*, 178(1):293-343, 2002.

[Luc04a] S. Lucas. MU-TERM: A Tool for Proving Termination of Context-Sensitive Rewriting In *Proc. of RTA'04*, LNCS 3091:200-209, Springer-Verlag, Berlin, 2004. Available at http://www.dsic.upv.es/~slucas/csr/termination/muterm.

[Luc04b] S. Lucas. Polynomials for proving termination of context-sensitive rewriting. In *Proc. of FOSSACS'04*, LNCS 2987:318-332, Springer-Verlag, Berlin 2004.

[Luc05] S. Lucas. Polynomials over the reals in proofs of termination: from theory to practice. *RAIRO Theoretical Informatics and Applications*, 39(3):547-586, 2005.

[Luc06] S. Lucas. Proving termination of context-sensitive rewriting by transformation. *Information and Computation*, to appear 2006.

[Ohl02] E. Ohlebusch. *Advanced Topics in Term Rewriting.* Springer-Verlag, Berlin, 2002.

[Ter03] TeReSe, editor, *Term Rewriting Systems*, Cambridge University Press, 2003.

[Zan97] H. Zantema. Termination of Context-Sensitive Rewriting. In *Proc. of RTA'97*, LNCS 1232:172-186, Springer-Verlag, Berlin, 1997.

On Reduction Criteria for Probabilistic Reward Models

Marcus Größer[1], Gethin Norman[2], Christel Baier[1],
Frank Ciesinski[1], Marta Kwiatkowska[2], and David Parker[2]

[1] Universität Bonn, Institut für Informatik I, Germany
{baier, ciesinsk}@cs.uni-bonn.de, groesser@tcs.inf.tu-dresden.de
[2] University of Birmingham, School of Computer Science, Edgbaston, United Kingdom
{kwiatkowska, norman, parker}@cs.bham.ac.uk

Abstract. In recent papers, the partial order reduction approach has been adapted to reason about the probabilities for temporal properties in concurrent systems with probabilistic behaviours. This paper extends these results by presenting reduction criteria for a probabilistic branching time logic that allows specification of constraints on quantitative measures given by a reward or cost function for the actions of the system.

1 Introduction

Partial order reduction [13,25,32] is one of the most prominent techniques for tackling the state explosion problem for concurrent software systems. It has been implemented in many tools and successfully applied to a large number of case studies, see e.g. [17]. Recently, the ample-set method [24] has been extended for concurrent probabilistic systems, both in the setting of quantitative linear time [5,7] and branching time [4] properties. The underlying models used in this work are Markov decision processes (MDPs), an extension of transition systems where nondeterminism can be used e.g. to model the interleaving of concurrent activities, to represent the interface with an unknown system environment or for abstraction purposes, and where probability serves e.g. to model coin tossing actions or to specify the frequency of exceptional (faulty) behaviour (such as losing messages from a buffer). Thus, MDPs arise as natural operational models for randomized distributed algorithms and communication or security protocols and are widely used in model checking. Equipped with reward or cost functions MDPs are also standard models in many other areas, such as operations research, reinforcement learning and robot path planning. In those fields a lot work has already been done on reducing the state (and/or actions) space via aggregating states (and/or actions) [2,29,11]. Opposed to many results in the field of machine learning that yield only approximations to optimal solutions, the results in the field of model checking offer some work on exact process equivalences, like (weak) bisimulation. Contrary to those approaches that rely on partition refinement and need global knowledge of the state space, the approach with partial order reduction can be implemented with local conditions and therefore be intertwined with the state space search on-the-fly, provided an appropriate high-level representation of the system is given.

The contribution of this paper is reduction criteria which are shown to be sound for an extension of probabilistic computation tree logic (PCTL) [6] that serves to reason about

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 309–320, 2006.

rewards or costs. Our logic, called PCTL$_r$, essentially agrees with the logic suggested by de Alfaro [9,8]. (PCTL$_r$ is also similar to the logic PRCTL [1,23] which relies on a Markov chain semantics, while PCTL$_r$-formulae are interpreted over MDPs.) PCTL$_r$ allows specifications regarding e.g. the packet loss characteristics of a queueing system, the energy consumption, or the average number of unsuccessful attempts to find a leader in a distributed system. We first explain how the ample-set conditions suggested in [4] for PCTL can be modified to treat reward-based properties specified in PCTL$_r$ and then identify a fragment of PCTL$_r$ (which still contains a wide range of non-trivial reward properties) where the weaker criteria of [4] are sufficient. We also present results on a new logic PCTL$_c$, that treats the rewards with a discounting semantics. As in the case of previous publications on partial order reduction for probabilistic systems, the major difficulty was to provide the proof of correctness. The general proof technique follows the line of [12,4] by establishing a bisimulation between the full and the reduced system. However, we depart here from these approaches by introducing a new variant of bisimulation equivalence for MDPs which borrows ideas from [21,31] and relies on the concept of norm functions [22,14]. This new type of bisimulation equivalence preserves PCTL$_r$-properties and might be useful also for other purposes.

Organization of the paper. Section 2 summarizes the basic definitions concerning Markov decision processes, reward structures and PCTL$_r$. Section 2 also recalls the partial order reduction approach for MDPs without reward structure and PCTL of [4] which we then extend to reason about rewards in Section 3. Section 4 identifies a class of reward-based properties that are preserved when using the weaker conditions of [4]. In Section 5 we discuss our approach in the setting of discounted rewards and Section 6 concludes the paper.

2 Preliminaries

Markov decision processes (MDPs), see e.g. [27]. An MDP is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, s_{\mathrm{init}}, \mathsf{AP}, L, \mathsf{rew})$ where S is a finite state space, $s_{\mathrm{init}} \in S$ is the initial state, Act a finite set of actions, AP a set of atomic propositions, $L : S \longrightarrow 2^{\mathsf{AP}}$ a labelling function, $\mathbf{P} : S \times Act \times S \longrightarrow [0,1]$ the three-dimensional transition probability matrix such that $\sum_{u \in S} \mathbf{P}(s, \alpha, u) \in \{0,1\}$ for all states s and actions α, and a function rew that assigns to each action $\alpha \in Act$ a reward $\mathsf{rew}(\alpha) \in \mathbb{R}$.

Action α is called *enabled* in state s if $\sum_{u \in S} \mathbf{P}(s, \alpha, u) = 1$. We write $Act(s)$ for the set of actions that are enabled in s. The states t with $\mathbf{P}(s, \alpha, t) > 0$ are called α-successors of s. For technical reasons, we require that $Act(s) \neq \emptyset$ for all states s. Action α is called a *stutter action* iff for all $s \in S$ where α is enabled in s, $L(s) = L(u)$ for all α-successors u of s. That is, stutter actions do not change the state labelling. Action α is called *non-probabilistic* iff for all states s, there is at most one α-successor. That is, if α is enabled in s then there is a state s_α with $\mathbf{P}(s, \alpha, s_\alpha) = 1$, while $\mathbf{P}(s, \alpha, u) = 0$ for all other states u. In particular, if $\alpha \in Act(s)$ is a non-probabilistic stutter action then $L(s) = L(s_\alpha)$.

An infinite *path* in an MDP is a sequence $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$ such that $\alpha_i \in Act(s_{i-1})$ and $\mathbf{P}(s_{i-1}, \alpha_i, s_i) > 0$ for all $i \geq 1$. We denote by *first*$(\varsigma) = s_0$ the starting state of ς and write *state*(ς, i) for the $(i+1)$th state in ς and $\rho(\varsigma, i)$ for the cumulative reward obtained through the first i actions. That is, if ς is as above then $state(\varsigma,i) = s_i$ and $\rho(\varsigma,i) = \mathsf{rew}(\alpha_1 \ldots \alpha_i)$ where $\mathsf{rew}(\alpha_1 \ldots \alpha_i) = \mathsf{rew}(\alpha_1)+\cdots+\mathsf{rew}(\alpha_i)$. If $T \subseteq S$ is a set of states then $\mathsf{Rew}(\varsigma,T)$ denotes the reward that is earned until a T-state is visited the first time. Formally, if $state(\varsigma,i) \in T$ and $state(\varsigma,j) \notin T$ for all $j < i$ then $\mathsf{Rew}(\varsigma,T) = \rho(\varsigma,i)$. If $state(\varsigma,i) \notin T$ for all $i \geq 0$ we set $\mathsf{Rew}(\varsigma,T) = \infty$. Finite paths (denoted by σ) are finite prefixes of infinite paths that end in a state. We use the notations $first(\sigma)$, $state(\sigma,i)$ and $\rho(\sigma,i)$ as for infinite paths and $|\sigma|$ for the length (number of actions). $Paths_{fin}(s)$ (resp. $Paths_{\omega}(s)$) denotes the set of all finite (resp. infinite) paths of $\mathcal{M}$ with $first(\cdot) = s$. Given a path $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$ we denote by $trace(\varsigma) = L(s_0), L(s_1), L(s_2), \ldots$ the word over the alphabet 2^{AP} obtained by the projection of ς to the state labels. Two infinite paths ς_1 and ς_2 in an MDP are called *stutter equivalent* iff there is an infinite word $\ell_1, \ell_2, \ldots$ over the alphabet 2^{AP} such that $trace(\varsigma_1) = \ell_1^{k_1}, \ell_2^{k_2}, \ldots$ and $trace(\varsigma_2) = \ell_1^{n_1}, \ell_2^{n_2}, \ldots$ where k_i, $n_i \geq 1$.

A *scheduler*, also often called policy, strategy or adversary, denotes an instance that resolves the nondeterminism in the states, and thus yields a Markov chain and a probability measure on the paths. We shall use here *history-dependent randomized schedulers* in the classification of [27]. They are defined as functions A that take as input a finite path σ and return a distribution over the actions $\alpha \in Act(last(\sigma))$.[1] A scheduler A is called deterministic if it chooses a unique action (with probability 1) for all finite paths. An A-path denotes an infinite or finite path σ that can be generated by A. Given a state s and a scheduler A, the behaviour of $\mathcal{M}$ under A can be formalised by a (possibly infinite-state) Markov chain. $\Pr^{A,s}$ denotes the standard probability measure on the Borel field of the infinite A-paths ς with $first(\varsigma) = s$. If $T \subseteq S$ then $\mathbb{E}^{A,s}(\lozenge T)$ denotes the expected value under A with starting state s for the random function $\varsigma \mapsto \mathsf{Rew}(\varsigma,T)$. Recall that $\mathsf{Rew}(\varsigma,T)$ denotes the reward that is earned by the prefix of ς that leads from the starting state s to a state in T and that $\mathsf{Rew}(\varsigma,T)$ equals ∞ if ς does not reach T. Thus, if there is a positive probability of not reaching T under scheduler A (from state s), then $\mathbb{E}^{A,s}(\lozenge T) = \infty$. If $s = s_{\text{init}}$ we simply write $\Pr^A$ and $\mathbb{E}^A$.

Probabilistic computation tree logic. PCTL is a probabilistic variant of CTL which has been introduced first for Markov chains [15] and then for Markovian models with nondeterminism [6,31]. We follow here the approach of de Alfaro [9,8] and extend PCTL with an operator $\mathcal{R}$ to reason about expected rewards. As partial order reduction relies on identifying stutter equivalent paths which might be distinguishable by the next step operator, we do not include the next step operator in the logic. PCTL$_r$-state formulae are therefore given by the grammar:

$$\Phi ::= \mathsf{true} \bigm| \mathsf{a} \bigm|\Phi \wedge \Phi\bigm| \neg\Phi \bigm| \mathcal{P}_J(\Phi_1 U_I \Phi_2) \bigm| \mathcal{R}_I(\Phi)$$

Here, $\mathsf{a} \in \mathsf{AP}$ is an atomic proposition, $J \subseteq [0,1]$ is a probability interval and $I \subseteq \mathbb{R} \cup \{-\infty,\infty\}$ a reward interval. We refer to the terms $\Phi_1 U_I \Phi_2$ as PCTL$_r$-path formulae. U_I denotes the standard until operator with a reward bound. The meaning of the path formula $\varphi = \Phi_1 U_I \Phi_2$ is that a Φ_2-state will be reached via a finite path σ where

[1] By a distribution on a finite set X we mean a function $\nu : X \to [0,1]$ such that $\sum_{x \in X} \nu(x) = 1$ and refer to $\nu(x)$ as the probability for x.

the cumulative reward is in I, while all states in σ, possibly except the last one, fulfil Φ_1. The state formula $\mathcal{P}_J(\varphi)$ holds for state s if for each scheduler A the probability measure of all infinite paths starting in s and fulfilling the path formula φ meets the probability bound given by J. On the other hand, $\mathcal{R}_I(\Phi)$ asserts that for any scheduler A the expected reward that is earned until a Φ-state has been reached meets the reward bound given by I. For instance, $\mathcal{R}_{[0,17]}(\mathsf{goal})$ asserts that independent of the scheduling policy the average costs to reach a goal state do not exceed 17. The formula $\mathcal{P}_{(0.9,1]}(\mathsf{true}\ U_{[0,4]}\ \mathsf{delivered})$ requires that the probability of a message being delivered with at most 4 retransmissions is greater than 0.9.

If $\mathcal{M}$ is an MDP and s a state in $\mathcal{M}$ then we write $s \models \Phi$ to denote that state-formula Φ holds in state s, and similarly, $\varsigma \models \varphi$ to denote that path formula φ holds for the infinite path ς. The formal semantics of the propositional logic fragment is standard and the semantics of the $\mathcal{P}$- and $\mathcal{R}$-operator is formalised by :

$$\begin{aligned} s \models \mathcal{P}_J(\Phi_1 U_I \Phi_2) &\Leftrightarrow \text{for all schedulers } A{:}\, \Pr^{A,s}\big\{\varsigma \in Paths_\omega(s) : \varsigma \models \Phi_1 U_I \Phi_2\big\} \in J \\ s \models \mathcal{R}_I(\Phi) &\Leftrightarrow \text{for all schedulers } A{:}\, \mathbb{E}^{A,s}(\Diamond Sat(\Phi)) \in I \end{aligned}$$

If $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$ then $\varsigma \models \Phi_1 U_I \Phi_2$ iff $\exists i \geq 0$ s.t. $s_i \models \Phi_2 \ \wedge\ \rho(\varsigma, i) \in I \ \wedge\ \forall j < i.\ s_j \models \Phi_1$. The satisfaction set of Φ in $\mathcal{M}$ is $Sat(\Phi) = \big\{s \in S : s \models \Phi\big\}$. State formula Φ is said to hold for an MDP if the initial state satisfies Φ.

Note that one could also give the $\mathcal{R}_I$ operator a different semantics as follows. $s \models \mathcal{R}_I(\Phi)$ if and only if for all schedulers A, such that the probability to reach $Sat(\Phi)$ from s equals 1, it holds that $\mathbb{E}^{A,s}(\Diamond Sat(\Phi)) \in I$. But this is irrelevant for our purposes.

Derived operators. Other Boolean connectives, such as disjunction $\vee$, implication $\rightarrow$, can be derived as usual. The temporal operator eventually $\Diamond$ is obtained in the standard way by $\Diamond_I \Phi = \mathsf{true}\ U_I \Phi$. The always-operator can be derived as in PCTL by the duality of lower and upper probability bounds. For the trivial reward-interval $I = (-\infty, \infty)$, we obtain the standard eventually, always and until operator. We simply write U, $\Diamond$ and $\Box$ rather than $U_{(-\infty,\infty)}$, $\Diamond_{(-\infty,\infty)}$ and $\Box_{(-\infty,\infty)}$, respectively.

PCTL denotes the sublogic of PCTL$_r$ that does not use the $\mathcal{R}$-operator and where the path-formulae have the trivial reward interval. Since the reward structure is irrelevant for PCTL-formulae, they can be interpreted over MDPs without reward structure.

The ample set method for PCTL [4]. Before presenting the partial order reduction citeria for PCTL$_r$ in Section 3, we briefly summarize the results of [4] for applying the ample-set method to PCTL model checking. The starting point is an MDP $\mathcal{M} = (S, Act, \mathbf{P}, s_{\text{init}}, \mathsf{AP}, L)$, without reward structure, to be verified against a PCTL-formula. Following Peled's ample-set method [24], the idea is to assign to any reachable state s a nonempty action-set $ample(s) \subseteq Act(s)$ and to construct a reduced MDP $\hat{\mathcal{M}}$ by using the action-sets $ample(s)$ instead of $Act(s)$. Formally, given a function $ample : S \rightarrow 2^{Act}$ with $\emptyset \neq ample(s) \subseteq Act(s)$ for all states s, the state space of the reduced MDP $\hat{\mathcal{M}} = (\hat{S}, Act, \hat{\mathbf{P}}, s_{\text{init}}, \mathsf{AP}, \hat{L})$ induced by $ample$ is the smallest set $\hat{S} \subseteq S$ that contains s_{init} and any state u where $\mathbf{P}(s, \alpha, u) > 0$ for some $s \in \hat{S}$ and $\alpha \in ample(s)$. The labelling function $\hat{L} : \hat{S} \rightarrow 2^{\mathsf{AP}}$ is the restriction of the original labelling function L to the state-set $\hat{S}$. The transition probability matrix of $\hat{\mathcal{M}}$ is given by $\hat{\mathbf{P}}(s, \alpha, t) = \mathbf{P}(s, \alpha, t)$ if $\alpha \in ample(s)$ and 0 otherwise. State s is called fully expanded if $ample(s) = Act(s)$.

A1 **(Stutter-condition)** If $ample(s) \neq Act(s)$ then all actions $\alpha \in ample(s)$ are stutter actions.

A2 **(Dependence-condition)** For each path $\sigma = s \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} s_n \xrightarrow{\gamma} \cdots$ in $\mathcal{M}$ where γ is dependent on $ample(s)$ there exists an index $i \in \{1, \ldots, n\}$ such that $\alpha_i \in ample(s)$.

A3 **(Cycle-condition)** On each cycle $s \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n = s$ in $\hat{\mathcal{M}}$ there exists a state s_i which is fully expanded, i.e., $ample(s_i) = Act(s_i)$.

A4 **(Branching condition)** If $ample(s) \neq Act(s)$ then $ample(s)$ is a singleton consisting of a non-probabilistic action.

Fig. 1. Conditions for the ample-set method for PCTL [4]

The main ingredient of any partial order reduction technique in the non-probabilistic or probabilistic setting is an adequate notion for the independence of actions. The definition for the independence of actions α and β in the composed transition system (which captures the semantics of the parallel composition of all processes that run in parallel) relies on recovering the interleaving 'diamonds'. Formally, two distinct actions α and β are called *independent* (in $\mathcal{M}$) iff for all states $s \in S$ with $\{\alpha, \beta\} \subseteq Act(s)$, (I1) $\alpha \in Act(u)$ for each β-successor u of s, (I2) $\beta \in Act(u)$ for each α-successor u of s, and (I3) $\mathbf{P}(s, \alpha\beta, w) = \mathbf{P}(s, \beta\alpha, w)$ for all $w \in S$ where $\mathbf{P}(s, \gamma\delta, w) = \sum_{u \in S} \mathbf{P}(s, \gamma, u) \cdot \mathbf{P}(u, \delta, w)$ for $\gamma, \delta \in Act$. Two different actions α and β are called *dependent* iff α and β are not independent. If $D \subseteq Act$ and $\alpha \in Act \setminus D$ then α is called independent of D iff for all actions $\beta \in D$, α and β are independent. Otherwise, α is called dependent on D.

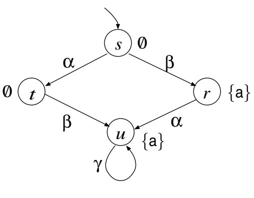
To preserve PCTL properties, [4] use the four conditions in Fig. 1. These rely on a slight modification of the conditions by Gerth et al [12] for preserving CTL-properties and can be implemented in an on-the-fly state space exploration [25,3].

Theorem 1 **([4]).** *If (A1)-(A4) hold then $\mathcal{M}$ and $\hat{\mathcal{M}}$ fulfil the same* PCTL*-formulae.*

3 Reduction Criteria for Rewards

In the sequel, we assume that we are given an MDP $\mathcal{M}$ and discuss the partial order reduction approach for properties specified in PCTL$_r$. We first show that (A1)-(A4) are not sufficient to preserve PCTL$_r$ properties with nontrivial reward bounds. To treat full PCTL$_r$, we shall need a modification of the branching condition (A4).

Example 1. We begin with a simple example illustrating that (A1)-(A4) cannot ensure that all PCTL$_r$-formulae are preserved. Consider the following MDP with the actions α, β, γ that are all non-probabilistic and where $\mathsf{rew}(\alpha) = \mathsf{rew}(\beta) = \mathsf{rew}(\gamma) = 1$.



Since α and β are independent and α is a stutter action, (A1)-(A4) allow for a reduction obtained through $ample(s) = \{\alpha\}$. Thus, $\hat{S} = \{s, t, u\}$. Consider the PCTL$_r$ formula $\Phi = \mathcal{R}_{[2,\infty)}(\mathsf{a})$. Then, the reduced system $\hat{\mathcal{M}}$ satisfies Φ, while the original system $\mathcal{M}$ does not, because $\mathcal{M}$ might choose action β in s which yields the expected reward 1 to reach an a-state. □

We now discuss how to strengthen conditions (A1)-(A4) such that reward-based properties are preserved. We start with some simple observations. First, as $\hat{\mathcal{M}}$ is a sub-MDP of the original system $\mathcal{M}$, any scheduler A for $\hat{\mathcal{M}}$ is also a scheduler for $\mathcal{M}$. Thus:

Lemma 1. *Let Φ_1, Φ_2 be PCTL$_r$-formulae with $Sat_{\mathcal{M}}(\Phi_i) \cap \hat{S} = Sat_{\hat{\mathcal{M}}}(\Phi_i)$, $i = 1,2$.*

(i) $\mathcal{M} \models \mathcal{R}_I(\Phi_1) \quad\Rightarrow\; \hat{\mathcal{M}} \models \mathcal{R}_I(\Phi_1)$,
(ii) $\mathcal{M} \models \mathcal{P}_J(\Phi_1 U_I \Phi_2) \Rightarrow\; \hat{\mathcal{M}} \models \mathcal{P}_J(\Phi_1 U_I \Phi_2)$.

The converse directions in Lemma 1 do not hold in general as $\mathcal{M}$ might have "more" schedulers than $\hat{\mathcal{M}}$. To get a feeling of how to modify the reduction criteria for PCTL$_r$, let us first give some informal explanations. In [4], the soundness proof of (A1)-(A4) for PCTL establishes a kind of bisimulation between the full MDP $\mathcal{M}$ and the reduced MDP $\hat{\mathcal{M}}$ which allows one to transform any scheduler A for $\mathcal{M}$ into a scheduler B for $\hat{\mathcal{M}}$ such that A and B yield the same probabilities for PCTL-path formulae. As in the case of the ample-set method for verifying linear time properties (where (A1)-(A3) and a weaker form of (A4) are sufficient [5,7]) this scheduler-transformation yields a transformation of the A-paths into "corresponding" B-paths. Let us look at this path-transformation "path ς in $\mathcal{M} \rightsquigarrow$ path $\hat{\varsigma}$ in $\hat{\mathcal{M}}$" which, in fact, is already known from the non-probabilistic case [24]. The path $\hat{\varsigma}$ in $\hat{\mathcal{M}}$ is obtained through a sequence of paths $\varsigma_0, \varsigma_1, \varsigma_2, \ldots$ in $\mathcal{M}$ such that the first i-steps in ς_i and ς_{i+1} agree and are composed of transitions in $\hat{\mathcal{M}}$. The switch from ς_i to ς_{i+1} is performed as follows.

Let $\pi = s_1 \xrightarrow{\alpha_1} s_2 \xrightarrow{\alpha_2} \cdots$ be the suffix of ς_i starting with the $(i{+}1)$th step (by the above, s_1 is a state in $\hat{\mathcal{M}}$). Our goal is to construct a stutter equivalent path $\hat{\pi}$ from s_1 that starts with an action in $ample(s_1)$. We then may compose the prefix of ς_i from $first(\varsigma_i)$ to s_1 with $\hat{\pi}$ to obtain the path ς_{i+1}. If $\alpha_1 \in ample(s_1)$ then we may put $\pi = \hat{\pi}$. Let us now assume that $\alpha_1 \notin ample(s_1)$. Then, by (A4), $ample(s_1)$ consists of a single non-probabilistic action.

(T1) If there is some index $j \geq 2$ such that $\alpha_j \in ample(s_1)$ then choose the smallest such index j and replace the action sequence $\alpha_1 \ldots \alpha_{j-1}\alpha_j\alpha_{j+1} \ldots$ with $\alpha_j\alpha_1 \ldots \alpha_{j-1}\ \alpha_{j+1} \ldots$. This is possible since by (A2) the actions $\alpha_1, \ldots, \alpha_{j-1}$ are independent of α_j. The resulting path $\hat{\pi}$ is stutter-equivalent to π by condition (A1).

(T2) If $\alpha_j \notin ample(s_1)$ for all $j \geq 1$ and $ample(s_1) = \{\beta\}$ then replace the action sequence $\alpha_1\alpha_2 \ldots$ with $\beta\alpha_1\alpha_2 \ldots$. Again, (A2) ensures that each α_j is independent of β. (A1) yields the stutter-equivalence of π and the resulting path $\hat{\pi}$.

Note, that the insertion of the additional action in transformation (T2) possibly changes the cumulative reward. Since we are interested in the cumulative reward that is gained until a certain state labelling is reached, the action permutation in transformation (T1) possibly changes this reward, as can be seen in Example 1 (note that a stutter action is permuted to the front of the action sequence).

To establish the equivalence of $\mathcal{M}$ and $\hat{\mathcal{M}}$ for PCTL$_r$ it seems to be sufficient to ensure that, in transformation (T2), the additional action β has zero reward, and in transformation (T1), the stutter action α_j, that is permuted to the front of the action sequence, has zero reward. This motivates the following stronger branching condition:

A4′ (New branching condition) If $ample(s) \neq Act(s)$ then $ample(s) = \{\beta\}$ for some non-probabilistic action with $\mathrm{rew}(\beta) = 0$.

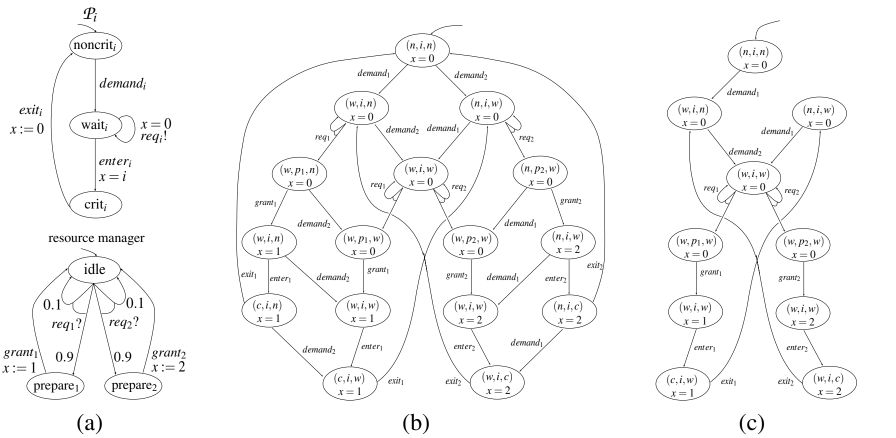


Fig. 2. Mutual exclusion example: (a) components, (b) full system and (c) reduced MDP

Theorem 2. *If (A1)-(A3), (A4′) hold then $\mathcal{M}$ and $\hat{\mathcal{M}}$ satisfy the same PCTL$_r$ formulae.*

Example 2. To illustrate our approach we consider a simple mutual exclusion protocol in which the processes P_1 and P_2 attempt to access a common resource controlled by a resource manager. A shared variable x is used to guarantee mutual exclusion and we assume that the communication is unreliable (requests to the resource manager are corrupted/lost with probability 0.1). Fig. 2(a) presents the different components of the system. Associating a reward of 1 with the actions req_1 and req_2 and 0 with all other actions, using PCTL$_r$ one can, for example, specify:

- $R_{\leq 1.4}(\mathsf{crit}_1 \vee \mathsf{crit}_2)$: the expected number of requests before a process enters the critical section is at most 1.4;
- $\mathcal{P}_{>0.7}(\mathsf{true}\ U_{[0,6]}\ \mathsf{crit}_1 \vee \mathsf{crit}_2)$: the probability that a process enters its critical section after at most 6 requests have been issued is strictly greater than 0.7.

Fig. 2(b) gives the full MDP for the system and (assuming $\mathsf{AP} = \{\mathrm{crit}_1, \mathrm{crit}_2\}$) one can construct the reduced system given in Fig. 2(c) satisfying conditions (A1)-(A4′). □

4 Preservation Result for (A1)-(A4) and Reward-Based Properties

We now turn to the question of which properties with nontrivial reward bounds are preserved by (A1)-(A3) and the original branching condition (A4) in Fig. 1. Let us again look at the path transformation described in (T1) and (T2) where, given a path π in $\mathcal{M}$ a path $\hat{\pi}$ is generated, where either the action sequence of $\hat{\pi}$ is a permutation of the action sequence of π (T1) or $\hat{\pi}$ starts with a non-probabilistic stutter action and then performs the same action sequence as the original path π (T2). As the rewards are in $\mathbb{R}$ we do not know, how the cumulative reward of $\hat{\pi}$ has changed compared to that of π. If we however require that the rewards of all actions are *non-negative*, along the modified path $\hat{\pi}$ a reward equal or greater will be earned than that along π. This yields an informal explanation why the additional power of $\mathcal{M}$ can lead to smaller minimal expected rewards, but the maximal expected rewards agree in $\mathcal{M}$ and $\hat{\mathcal{M}}$. Similarly, we might expect that the minimal probabilities for events of the form $\mathsf{a}_1 U_{[0,r]} \mathsf{a}_2$ agree under $\mathcal{M}$ and $\hat{\mathcal{M}}$. The same holds for maximal probabilities for events of the form $\Box_{[0,r]}\mathsf{a}$. This motivates the definition of the following sublogic of PCTL$_r$.

Let PCTL$_r^-$ be the sublogic of PCTL$_r$ which only uses the $\mathcal{R}$-operator with upper reward bounds, i.e., formulae of the form $\mathcal{R}_{[0,r]}(\Phi)$, and where the probabilistic operator is only used in combination with PCTL-path formulae $\Phi_1 U \Phi_2$ or with the until-operator in combination with upper reward and lower probability bounds or in combination with lower reward and upper probability bounds or with the always-operator in combination with upper reward and upper probability bounds or in combination with lower reward and lower probability bounds, e.g. $\mathcal{P}_{[0,p]}(\Box_{[0,r]}\Phi)$ or $\mathcal{P}_{(p,1]}(\Phi_1 U_{[0,r]} \Phi_2)$. Note that PCTL is contained in PCTL$_r^-$. (The result stated in Theorem 3 would still hold when dealing with a release- or weak until operator rather than the always-operator.)

Theorem 3. *If (A1)-(A4) hold and* $\mathsf{rew}(\alpha) \geq 0$ *for all* $\alpha \in Act$ *then* $\mathcal{M}$ *and* $\hat{\mathcal{M}}$ *satisfy the same* PCTL$_r^-$ *formulae.*

Proof. (*sketch*) As is the case for many other types of (bi)simulation relations for probabilistic systems, our notion of bisimulation equivalence will use the concept of *weight functions* [18,19]. Let S, S' be finite sets and $R \subseteq S \times S'$. If ν and ν' are distributions on S and S' respectively then a weight function for (ν,ν') with respect to R denotes a function $w : S \times S' \to [0,1]$ such that $\{(s,s') : w(s,s') > 0\} \subseteq R$, $\sum_{u' \in S'} w(s,u') = \nu(s)$ and $\sum_{u \in S} w(u,s') = \nu'(s')$ for all $s \in S$, $s' \in S'$. We write $\nu \sqsubseteq_R \nu'$ iff there exists a weight function for (ν,ν') with respect to R and refer to $\sqsubseteq_R$ as the lifting of R to distributions.

Definition 1 (Normed (bi)simulation). Let $\mathcal{M} = (S_{\mathcal{M}}, Act, \mathbf{P}_{\mathcal{M}}, s^{\mathcal{M}}_{\text{init}}, \mathsf{AP}, L_{\mathcal{M}}, \mathsf{rew})$ and $\mathcal{N} = (S_{\mathcal{N}}, Act, \mathbf{P}_{\mathcal{N}}, s^{\mathcal{N}}_{\text{init}}, \mathsf{AP}, L_{\mathcal{N}}, \mathsf{rew})$ be two MDPs with the same set of atomic propositions, the same action set Act and the same reward structure $\mathsf{rew} : Act \to I\!R_{\geq 0}$. A normed reward simulation for $(\mathcal{M},\mathcal{N})$ with respect to rew is a triple (R, η_1, η_2) consisting of a binary relation $R \subseteq S_{\mathcal{M}} \times S_{\mathcal{N}}$ and functions $\eta_1, \eta_2 : R \to I\!N$ such that $(s^{\mathcal{M}}_{\text{init}}, s^{\mathcal{N}}_{\text{init}}) \in R$ and for each pair $(s,s') \in R$ the following conditions hold.

(N1) $L_{\mathcal{M}}(s) = L_{\mathcal{N}}(s')$

(N2) If $\alpha \in Act_{\mathcal{M}}(s)$ then at least one of the following conditions holds:

(N2.1) α is enabled in s' (i.e., $\alpha \in Act_{\mathcal{N}}(s')$) and $\mathbf{P}_{\mathcal{M}}(s,\alpha,\cdot) \sqsubseteq_R \mathbf{P}_{\mathcal{N}}(s',\alpha,\cdot)$,

(N2.2) α is a non-probabilistic stutter action s. th. $(s_\alpha, s') \in R$ and $\eta_1(s_\alpha,s') < \eta_1(s,s')$.

(N2.3) There is a non-probabilistic stutter action $\beta \in Act_{\mathcal{N}}(s')$ with $(s, s'_\beta) \in R$ and $\eta_2(s,s'_\beta) < \eta_2(s,s')$.

A normed bisimulation for $(\mathcal{M},\mathcal{N})$ is a tuple $(R,\eta_1,\eta_2,\eta_1^-,\eta_2^-)$ such that (R,η_1,η_2) and $(R^{-1},\eta_1^-,\eta_2^-)$ are normed simulations for $(\mathcal{M},\mathcal{N})$, resp. $(\mathcal{N},\mathcal{M})$. □

We write $\mathcal{M} \approx_{nb} \mathcal{N}$ iff there exists a normed bisimulation for $\mathcal{M}$ and $\mathcal{N}$.

A forming path from s to $\hat{s}$ means a path $s = s_0 \xrightarrow{\beta_0} s_1 \xrightarrow{\beta_1} \cdots \xrightarrow{\beta_{n-1}} s_n = \hat{s}$ where $\beta_0, \ldots, \beta_{n-1}$ are non-probabilistic stutter actions, and for $0 \leq i < n$, the singleton action-set $\{\beta_i\}$ fulfils the dependence condition (A2) for state s_i. A shortest forming path from s to $\hat{s}$ means a forming path from s to $\hat{s}$ where the cumulative reward is minimal under all forming paths from s to $\hat{s}$ and where the length (number of actions) is minimal under all forming paths with minimal cumulative reward. We will write $\mu(s,\hat{s})$ for the cumulative reward of all/some shortest forming path from s to $\hat{s}$. $s \rightsquigarrow \hat{s}$ denotes the existence of a forming path from s to $\hat{s}$ and we put $R = \{(s,\hat{s}) \in S \times \hat{S} : s \rightsquigarrow \hat{s}\}$.

If $(s,\hat{s}) \in R$ then

$$\mathrm{Pr}^{A,s}(\Pi(s, r{+}\mu(s,\hat{s}), C_1, \ldots, C_n)) \geq \mathrm{Pr}^{B,\hat{s}}(\Pi(\hat{s}, r, C_1, \ldots, C_n)) \qquad (*)$$

and $\mathrm{Pr}^{A,s}(\Pi(s, C_1, \ldots, C_n)) = \mathrm{Pr}^{B,\hat{s}}(\Pi(\hat{s}, C_1, \ldots, C_n))$. Here, we used the following notation. Let $u \in S$, $C_1, C_2, \ldots, C_n$ be a sequence of $\approx_{nb}$-equivalence classes with $C_i \neq C_{i+1}$ for $1 \leq i < n$ and $r \geq 0$. Then, $\Pi(u, r, C_1, \ldots, C_n)$ denotes the set of all infinite paths that have a finite prefix of the form $u_0 \rightarrow^*_{C_1} \tilde{u}_1 \xrightarrow{\gamma_1} u_2 \rightarrow^*_{C_2} \tilde{u}_2 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_{n-2}} u_{n-1} \rightarrow^*_{C_{n-1}} \tilde{u}_{n-1} \xrightarrow{\gamma_{n-1}} u_n$ where $u_0 = u$ and the total reward is $\leq r$ and $u_n \in C_n$. The actions γ_i are arbitrary. In this context, $v \rightarrow^*_C \tilde{v}$ means a finite path built out of non-probabilistic stutter actions such that v, $\tilde{v}$ and all intermediate states of that path belong to C. $\Pi(u, C_1, \ldots, C_n)$ stands for the union of the path-sets $\Pi(u, r, C_1, \ldots, C_n)$ for arbitrary $r \geq 0$. For $s = s_{\text{init}} = \hat{s}$ we have $\mu(s,\hat{s}) = \mu(s_{\text{init}}, s_{\text{init}}) = 0$.

The above yields that for each scheduler A for $\mathcal{M}$ there exists a scheduler B for $\hat{\mathcal{M}}$ such that $\mathrm{Pr}^A(\Pi(s_{\text{init}}, r, C_1, \ldots, C_n)) \geq \mathrm{Pr}^B(\Pi(s_{\text{init}}, r, C_1, \ldots, C_n))$ for all $r \geq 0$ and all $\approx_{nb}$-equivalence classes $C_1, \ldots, C_n$. From this we can derive that $\mathcal{M}$ and $\hat{\mathcal{M}}$ fulfil the same PCTL$_r^-$ formulae. □

Example 3. Let us return to Example 2 and redefine the rewards such that the only nonzero rewards are for actions *demand*$_1$ and *demand*$_2$ which have reward 1. Now, in this situation the reduced MDP in Fig. 2(c) can no longer be constructed using (A1)-(A4′). However, this construction is still possible under (A1)-(A4).

This is demonstrated by the fact that both the reduced and full MDP satisfy the PCTL$_r^-$ property $\mathcal{R}_{[0,2]}(\mathsf{crit}_1 \vee \mathsf{crit}_2)$ (the maximum expected number of processes that can attempt to enter the critical section before one of them does so is at most 2), while only the reduced model satisfies the PCTL$_r$ property $\mathcal{R}_{[2,\infty)}(\mathsf{crit}_1 \vee \mathsf{crit}_2)$ (the minimum expected number is at least 2). □

5 Reward Properties w.r.t Discounted Rewards

In many research areas (e.g. economics, operations research, control theory) rewards are treated with a different semantics, namely as so-called *discounted rewards* [27], where given a discount factor $0 < c < 1$, the reward of the i-th action of a path is multiplied with c^{i-1}. This interpretation of rewards reflects the fact that a reward (e.g. a payment) in the future is not worth quite as much as it is now (e.g. due to inflation). In this Section we investigate our partial order approach for discounted rewards.

Given a path $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$ and a discount factor $c \in (0,1)$, we denote by $\rho_c(\varsigma, i) = \mathsf{rew}_c(\alpha_1 \ldots \alpha_i) = c^0 \cdot \mathsf{rew}(\alpha_1) + c^1 \cdot \mathsf{rew}(\alpha_2) + \cdots + c^{i-1} \cdot \mathsf{rew}(\alpha_i)$ the cumulative discounted reward obtained through the first i actions.

With this on hand we can define the logic PCTL$_c$, which is a variant of PCTL$_r$. In PCTL$_c$, we use the new operators U_I^c and $\mathcal{R}_I^c$ instead of U_I and $\mathcal{R}_I$, where instead of the cumulative reward $\rho(\varsigma, i)$ the cumulative discounted reward $\rho_c(\varsigma, i)$ is used in the semantics of those new operators. The semantics of the U_I^c operator is as follows. Given a path $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$, we say that $\varsigma \models \Phi_1 U_I^c \Phi_2$ iff $\exists i \geq 0$ s.t $s_i \models \Phi_2 \wedge \forall j < i : s_j \models \Phi_1 \wedge \rho_c(\varsigma, i) \in I$. Similarly, given a set of states $T \subseteq S$ we denote by $\mathsf{Rew}_c(\varsigma, T)$ the discounted reward that is earned until a T-state is visited the first time. Formally, if $state(\varsigma, i) \notin T$ for all $i \geq 0$ then $\mathsf{Rew}_c(\varsigma, T) = \infty$. If $state(\varsigma, i) \in T$ and $state(\varsigma, j) \notin T$ for all $j < i$ then $\mathsf{Rew}_c(\varsigma, T) = \rho_c(\varsigma, i)$. For $T \subseteq S$ and a scheduler A, $\mathbb{E}_c^{A,s}(\Diamond T)$ denotes the expected value under A with starting state s for the random function $\varsigma \mapsto \mathsf{Rew}_c(\varsigma, T)$. Then $s \models \mathcal{R}_I^c(\Phi)$ iff $\forall$ schedulers A: $\mathbb{E}_c^{A,s}(\Diamond Sat(\Phi)) \in I$.

A simple example shows that theorem 2 does not hold for PCTL$_c$ (even if all rewards are nonnegative). Consider the MDP $\mathcal{M}$ in example 1 on page 313. We assign the following rewards : $\mathsf{rew}(\alpha) = 0, \mathsf{rew}(\beta) = \mathsf{rew}(\gamma) = 1$. Choosing $ample(s) = \{\alpha\}$, conditions (A1)-(A3) and (A4') are satisfied. However, if we consider the formula $\Phi = \mathcal{R}_{[0,c]}^c(a)$, we gain that the reduced system $\hat{\mathcal{M}}$ satisfies Φ while the original system $\mathcal{M}$ does not, because $\mathcal{M}$ might choose action β in state s which yields the expected discounted reward to reach an a-state to be $c^0 \cdot \mathsf{rew}(\beta) = 1 > c$.

The reader should notice that due to the discounting, the transformations (T1) and (T2) described in Section 3 on page 314 change the reward of a given path, even under condition (A4') which requires the ample set of a non-fully expanded state to be a singleton consisting of a non-probabilistic action with zero reward. Nevertheless, the following holds: given an MDP M with only *non-negative* rewards, ample-sets that satisfy (A1)-(A3) and (A4') and a path ς in $\mathcal{M}$, let $\hat{\varsigma}$ be a path that emanates from ς by applying transformation (T1) or (T2). Then $\rho_c(\hat{\varsigma}, i) \leq \rho_c(\varsigma, i)$. Similarly as in Section 4 this informally explains that the additional power of $\mathcal{M}$ can lead to greater maximal expected rewards, but the minimal expected rewards agree in $\mathcal{M}$ and $\hat{\mathcal{M}}$. Also, the maximal probabilities for events of the form $a_1 U_{[0,r]}^c a_2$ agree under $\mathcal{M}$ and $\hat{\mathcal{M}}$. This motivates the definition of the following sublogic of $PCTL_c$.

Let $PCTL_c^-$ be the sublogic of $PCTL_c$ which uses the $\mathcal{R}^c$ operator only with lower reward bounds (i.e $\mathcal{R}_{[r,\infty)}^c \Phi$) and where the probabilistic operator is only used in combination with PCTL-path formulae $\Phi_1 U \Phi_2$ or with the until-operator in combination with lower reward and lower probability bounds or in combination with upper reward and upper probability bounds or with the always-operator in combination with upper reward and lower probability bounds or in combination with lower reward and upper probability bounds, e.g. $\mathcal{P}_{[0,p]}(\Box_{[r,\infty)} \Phi)$ or $\mathcal{P}_{[0,p]}(\Phi_1 U_{[0,r]} \Phi_2)$. Note that PCTL is contained in PCTL$_c^-$.

Theorem 4. *If (A1)-(A3) and (A4') hold and* $\mathsf{rew}(\alpha) \geq 0$ *for all* $\alpha \in Act$ *then* $\mathcal{M}$ *and* $\hat{\mathcal{M}}$ *satisfy the same* PCTL$_c^-$ *formulae.*

6 Conclusion

The goal of this paper was to study the theoretical foundations of the ample-set approach for the logic PCTL$_r$, a variant of PCTL with reward-bounded temporal modalities and an expectation operator. The main results of this paper are that the ample-set conditions presented in [4] for PCTL preserve a class of non-trivial reward-based properties (Theorem 3) and that a slight modification of the conditions of [4] are sufficient to treat full PCTL$_r$ (Theorem 2). The proofs of these results have been established by means of a new notion of weak bisimulation for MDPs which preserves PCTL$_r$ and – since it is simpler than other notions of weak bisimulation equivalence for MDPs – might also be useful for other purposes. Moreover we investigated the logic PCTL$_c$, a variant of PCTL$_r$ where the rewards are given a discounting semantics. We presented ample-set conditions that preserve a non-trivial subset of PCTL$_c$ properties if all given rewards are non-negative (Theorem 4).

Besides being of theoretical interest, the results of this paper also have a practical impact. First experimental results on the ample set approach for MDPs (without reward structure) with the forthcoming model checker LiQuor [3] show that although the criteria needed for probabilistic systems are stronger than in the non-probabilistic case, good reductions can be obtained. Furthermore, the bottleneck in analysis of probabilistic systems modelled by MDPs are the required techniques for solving linear programs. Since the amount of time required for the construction of the reduced MDP is negligible compared to the running time of linear program solvers, even small reductions can increase the efficiency of the quantitative analysis.

In future work, we plan to integrate the partial order reduction techniques suggested here in the symbolic MTBDD-based model checker PRISM [16] by constructing a syntactic representation of the reduced MDP at compile time, in the style of static partial order reduction [20] which permits a combination of partial order reduction with symbolic BDD-based model checking.

References

1. S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *Proc. FORMATS*, volume 2791 of *Lecture Notes in Computer Science*, pages 88–104, 2003.
2. Mehran Asadi and Manfred Huber. Action dependent state space abstraction for hierarchical learning systems. In *Proc. IASTED*, Insbruck, Austria, 2005.
3. C. Baier, F. Ciesinski, and M. Groesser. Quantitative analysis of distributed randomized protocols. In *Proc.FMICS*, 2005.
4. C. Baier, P. D'Argenio, and M. Größer. Partial order reduction for probabilistic branching time. In *Proc. QAPL*, 2005.
5. C. Baier, M. Größer, and F. Ciesinski. Partial order reduction for probabilistic systems. In QEST 2004 [28], pages 230–239.
6. A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proc. FST & TCS*, volume 1026 of *LNCS*, pages 499–513, 1995.
7. P.R. D'Argenio and P. Niebert. Partial order reduction on concurrent probabilistic programs. In QEST 2004 [28], pages 240–249.
8. L. de Alfaro. *Formal Verification of Probabilistic Systems*. PhD thesis, Stanford University, Department of Computer Science, 1997.

9. L. de Alfaro. Temporal logics for the specification of performance and reliability. In *Proc. STACS*, volume 1200 of *Lecture Notes in Computer Science*, pages 165–179, 1997.
10. L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proc. 13th LICS*, IEEE Press, pages 454–465, 1998.
11. Thomas Dean, Robert Givan, and Sonia Leach. Model reduction techniques for computing approximately optimal solutions for markov decision processes. In *Proc. 13th UAI*, pages 124–131, San Francisco, California, 1997. Morgan Kaufmann Publishers.
12. R. Gerth, R. Kuiper, D. Peled, and W. Penczek. A partial order approach to branching time logic model checking. In *Proc. 3rd ISTCS'95*, pages 130–139. IEEE Press, 1995.
13. P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems: An Approach to the State Explosion Problem*, volume 1032 of *LNCS*. Springer-Verlag, 1996.
14. D. Griffioen and F. Vaandrager. Normed simulations. In *Proc. 10th International Computer Aided Verification Conference*, volume 1427 of *LNCS*, pages 332–344, 1998.
15. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
16. A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th TACAS*, 2006. To appear.
17. G. Holzmann. *The SPIN Model Checker, Primer and Reference Manual.* Addison Wesley, 2003.
18. C. Jones. *Probabilistic Non-Determinism.* PhD thesis, University of Edinburgh, 1990.
19. B. Jonsson and K. Larsen. Specification and refinement of probabilistic processes. In *Proc. LICS*, pages 266–277. IEEE CS Press, 1991.
20. R. Kurshan, V. Levin, M. Minea, D. Peled, and H. Yenign. Static partial order reduction. In *Proc. TACAS*, volume 1384 of *LNCS*, pages 345–357, 1998.
21. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
22. K. Namjoshi. A simple characterization of stuttering bisimulation. In *Proc. FSTTCS*, volume 1346 of *LNCS*, pages 284–296, 1997.
23. N. Pekergin and Sana Younes. Stochastic model checking with stochastic comparison. In *Proc. EPEW/WS-FM*, volume 3670 of *LNCS*, pages 109–123, 2005.
24. D. Peled. All from one, one for all: On model checking using representatives. In *Proc. 5th CAV*, volume 697 of *LNCS*, pages 409–423, 1993.
25. D. Peled. Partial order reduction: Linear and branching time logics and process algebras. In [26], pages 79–88, 1996.
26. D. Peled, V. Pratt, and G. Holzmann, editors. *Partial Order Methods in Verification*, volume 29(10) of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 1997.
27. M. L. Puterman. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., New York, 1994.
28. *Proceedings of the 1st International Conference on Quantitative Evaluation of SysTems (QEST 2004).* Enschede, the Netherlands. IEEE Computer Society Press, 2004.
29. Balaraman Ravindran and Andrew G. Barto. Model minimization in hierarchical reinforcement learning. In *Proc. SARA*, pages 196–211. LNCS, 2002.
30. R. Segala. *Modeling and Verification of Randomized Distributed Real-Time Systems.* PhD thesis, Massachusetts Institute of Technology, 1995.
31. R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–273, 1995.
32. A. Valmari. Stubborn set methods for process algebras. In [26], pages 79–88, 1996.

Distributed Synthesis for Well-Connected Architectures*

Paul Gastin[1], Nathalie Sznajder[1], and Marc Zeitoun[2]

[1] LSV, ENS de Cachan & CNRS
61, Av. du Président Wilson, F-94235 Cachan Cedex, France
{Paul.Gastin, Nathalie.Sznajder}@lsv.ens-cachan.fr
[2] LaBRI, Université Bordeaux 1 & CNRS
351, Cours de la Libération, F-33405 Talence Cedex, France
mz@labri.fr

Abstract. We study the synthesis problem for external linear or branching specifications and distributed, synchronous architectures with arbitrary delays on processes. *External* means that the specification only relates input and output variables. We introduce the subclass of uniformly well-connected (UWC) architectures for which there exists a routing allowing each output process to get the values of all inputs it is connected to, as soon as possible. We prove that the distributed synthesis problem is decidable on UWC architectures if and only if the set of all sets of input variables visible by output variables is totally ordered, under set inclusion. We also show that if we extend this class by letting the routing depend on the output process, then the previous decidability result fails. Finally, we provide a natural restriction on specifications under which the whole class of UWC architectures is decidable.

1 Introduction

Synthesis is an essential problem in computer science considered by Church in [2]. It consists in translating a system property, given in a high level specification language (such as temporal logic) into a low-level equivalent model (such as a finite automaton). The problem can be parametrized by the specification language and the target model. For instance, synthesis for infinite sequential systems from monadic second order formulas is simply Büchi's theorem.

In this paper, we address the synthesis problem for distributed open synchronous systems and temporal logic specifications. This specific question has been first studied in [11], where general synthesis has been proved undecidable for LTL specifications, and LTL synthesis for pipeline architectures has been shown non elementarily decidable, the lower bound following from a former result on multiplayer games [10]. For local specifications, constraining only variables local to processes [8], the general problem is undecidable (though doubly flanked pipelines become decidable.)

* Work partly supported by the European research project HPRN-CT-2002-00283 GAMES and by the ACI Sécurité Informatique 2003-22 (VERSYDIS).

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 321–332, 2006.

The pipeline architecture has been shown decidable for CTL* *full* specifications [5], that is, specifications allowed to constrain all variables of the system. In this case, where decidability of the distributed synthesis is obtained, full specifications strengthen the result.

A decision criterion, established in [3] for full specifications, implies that the architecture of Figure 1 is undecidable. The reason is that specifications are allowed to enforce a constant value on variable t, breaking the link between processes p_0 and p_1. For the undecidability part of the criterion, allowing specifications on all variables allows easy reductions to the basic undecidable architecture of Pnueli and Rosner [11], for instance by breaking communication links at will.

In the seminal paper [11], specifications were assumed to be *external*, or *input-output*: only variables communicating with the environment could be constrained. The way processes of the system communicate was only restricted by the communication architecture, not by the specification. This is very natural from a practical point of view: when writing a specification, we are only concerned by the input/output behavior of the system and we should leave to the implementation all freedom on its internal behavior. For that reason, solving the problem for external specifications is more relevant and useful - albeit more difficult - than a decidability criterion for arbitrary specifications. We will show that the architecture of Figure 1 is decidable for *external* specifications, that is, if we do not constrain the internal variable t.

Contributions. We consider the synthesis problem for synchronous semantics, where each process is assigned a nonnegative delay. The delays can be used to model latency in communications, or slow processes. This model has the same expressive power as the one where delays sit on communication channels, and it subsumes both the 0-delay and the 1-delay classical semantics [11,5].

To rule out unnatural properties yielding undecidability, the specifications we consider are external, coming back to the original framework of [11,2]. We first determine a sufficient condition for undecidability with external specifications, that generalizes the undecidability result of [11]. We next introduce *uniformly well-connected* (UWC) architectures. Informally, an architecture is UWC if there exists a routing of input variables allowing each output process to get, as soon as possible, the values of all inputs it is connected to. Using tree automata, we prove that for such architectures, the sufficient condition for undecidability becomes a criterion, for external specifications. We also propose a natural restriction on specifications for which all UWC architectures becomes decidable.

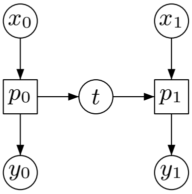


Fig. 1. Architecture decidable for external/undecidable for full specifications

Finally, we introduce the larger class of *well-connected architectures*, in which the routing of input variables to an output process can depend on that process. We show that our criterion is not necessary anymore for this larger class. Whether the restricted external specifications are always decidable for this class, as it is the case for UWC architectures, remains open. The undecidability proof highlights the surprising fact that in Figure 1, blanking out a *single* information bit in the transmission of x_0 to p_1 through t suffices to yield undecidability. This is a step forward in understanding decidability limits for distributed synthesis.

Due to lack of space, proofs are omitted or only sketched in this extended abstract. A full version is available in [4].

2 Preliminaries

Trees and tree automata. Given two finite sets X and Y, a Y-labeled (full) X-tree is a (total) function $t : X^* \to Y$ where elements of X are called directions, and elements of Y are called labels. A word $\sigma \in X^*$ defines a node of t and $t(\sigma)$ is its label. The empty word ε is the root of the tree. A word $\sigma \in X^\omega$ is a branch. In the following, a tree $t : X^* \to Y$ will be called an (X, Y)-tree.

A non-deterministic tree automaton (NDTA) $\mathfrak{A} = (X, Y, Q, q_0, \delta, \alpha)$ runs on (X, Y)-trees. It consists of a finite set of states Q, an initial state q_0, a transition function $\delta : Q \times Y \to \mathcal{P}(Q^X)$ and an acceptance condition $\alpha \subseteq Q^\omega$. A *run* ρ of such an automaton over a (X, Y)-tree t is a (X, Q)-tree ρ such that for all $\sigma \in X^*$, $(\rho(\sigma \cdot x))_{x \in X} \in \delta(\rho(\sigma), t(\sigma))$. A run tree is accepting if all its branches $s_1 s_2 \cdots$ are such that $\rho(\varepsilon)\rho(s_1)\rho(s_1 s_2) \cdots \in \alpha$. The specific acceptance condition chosen among the classical ones is not important in this paper.

Architectures. An *architecture* $\mathcal{A} = (V \uplus P, E, (S^v)_{v \in V}, s_0, (d_p)_{p \in P})$ is a finite directed acyclic bipartite graph, where $V \uplus P$ is the set of vertices, and $E \subseteq (V \times P) \cup (P \times V)$ is the set of edges, such that $|E^{-1}(v)| \leq 1$ for all $v \in V$. Elements of P will be called *processes* and elements of V *variables*. Intuitively, an edge $(v, p) \in V \times P$ means that process p can read variable v, and an edge $(p, v) \in P \times V$ means that p can write on v. Thus, $|E^{-1}(v)| \leq 1$ means that a variable v is written by at most one process. Input and output variables are defined, respectively, by $V_\mathrm{I} = \{v \in V \mid E^{-1}(v) = \emptyset\}$ and $V_\mathrm{O} = \{v \in V \mid E(v) = \emptyset\}$. Variables in $V \setminus (V_\mathrm{I} \cup V_\mathrm{O})$ will be called *internal*. We assume that no process is minimal or maximal in the graph. Each variable v ranges over a finite domain S^v, given with the architecture. The initial value of the variables is $s_0 = (s_0^v)_{v \in V} \in \prod_{v \in V} S^v$. We will consider that $|S^v| \geq 2$ for all $v \in V$. In fact, if not, such a variable would always have the same value, and could be ignored. It will be convenient in some proofs to assume that $\{0, 1\} \subseteq S^v$ and that $s_0^v = 0$ for all $v \in V$. Each process $p \in P$ is associated with a delay $d_p \in \mathbb{N}$ that corresponds to the time interval between the moment the process reads the variables $v \in E^{-1}(p)$ and the moment it will be able to write on its own output variables. Note that delay 0 is allowed. In the following, for $v \in V$, we will often write d_v for d_p where $E^{-1}(v) = \{p\}$.

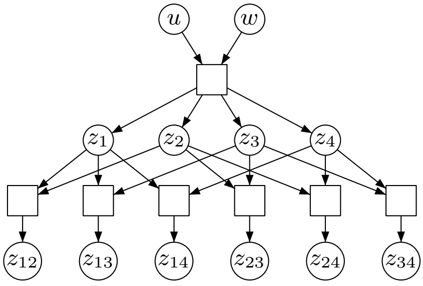


Fig. 2. An architecture

An example of an architecture is given in Figure 2, where processes are represented by boxes and variables by circles.

Runs. When $U \subseteq V$, S^U will denote $\prod_{v \in U} S^v$. A *configuration* of the architecture is given by a tuple $s \in S^V$ describing the value of the variables. For $s = (s^v)_{v \in V} \in S^V$, $U \subseteq V$, we denote by $s^U = (s^v)_{v \in U}$ the projection of the configuration s to the subset of variables U. A *run* of an architecture is an infinite sequence of configurations, *i.e.*, an infinite word over the alphabet S^V, starting with the initial configuration $s_0 \in S^V$ given by the architecture. If $\sigma = s_0 s_1 s_2 \cdots \in (S^V)^\omega$ is a run, then its projection on U is $\sigma^U = s_0^U s_1^U s_2^U \cdots$. Also, we denote by $\sigma[i]$ the prefix of length i of σ (by convention, $\sigma[i] = \varepsilon$ if $i \leq 0$). A *run tree* is a *full* tree $t : (S^{V_\mathrm{I}})^* \to S^V$, where $t(\varepsilon) = s_0$ and for $\rho \in (S^{V_\mathrm{I}})^*$, $r \in S^{V_\mathrm{I}}$, we have $(t(\rho \cdot r))^{V_\mathrm{I}} = r$. The projection of t on $U \subseteq V$ is the tree $t^U : (S^{V_\mathrm{I}})^* \to S^U$ defined by $t^U(\rho) = t(\rho)^U$.

Specifications. Specifications over a set $U \subseteq V$ of variables can be given, for instance, by a μ-calculus, CTL*, CTL, or LTL formula, with atomic propositions of the form $(v = a)$ for $v \in U$ and $a \in S^v$. We then say that the formula is in $\mathcal{L}(U)$ where $\mathcal{L}$ is the logic used. A specification is *external* if $U \subseteq V_\mathrm{I} \cup V_\mathrm{O}$. The validity of an external formula on a run tree t (or simply a run) only depends on its projection $t^{V_\mathrm{I} \cup V_\mathrm{O}}$ onto $V_\mathrm{I} \cup V_\mathrm{O}$.

Programs, strategies. We consider a discrete time, synchronous semantics. Informally, at step $i = 1, 2, \ldots$, the environment provides new values for input variables. Then, each process p reading values written by its predecessors or by the environment at step $i - d_p$, computes values for the variables it writes to, and writes them. Let $v \in V \setminus V_\mathrm{I}$ and let $R(v) = E^{-2}(v)$ be the set of variables read by the process writing to v. Intuitively, from a word $(s_0\sigma)^{R(v)}$ in $(S^{R(v)})^+$ representing the projection on $R(v)$ of some run prefix, a program (or a strategy) advices a value to write on variable v. But, since the process may have a certain delay d_v, the output of the strategy must not depend on the last d_v values of $(s_0\sigma)^{R(v)}$. Since all runs begin by s_0, this initial configuration is irrelevant for a strategy which only depends on $\sigma^{R(v)}$. Formally, a *program* (or *local strategy*) for variable v is a mapping $f^v : \left(S^{R(v)}\right)^+ \to S^v$ compatible with the delay d_v, i.e., such that for all $\rho, \rho' \in (S^{R(v)})^i$, if $\rho[i - d_v] = \rho'[i - d_v]$, then $f^v(\rho) = f^v(\rho')$. This condition ensures that the delay d_v is respected when computing the next value of variable v. A *distributed program* (or *distributed strategy*) is a tuple $F = (f^v)_{v \in V \setminus V_{\mathrm{I}}}$ of local strategies. A run $\sigma \in (S^V)^\omega$ is an *F-run* (or *F-compatible*) if for all $v \in V \setminus V_{\mathrm{I}}$, $s_i^v = f^v(\sigma^{R(v)}[i])$. Given an input sequence $\rho \in (S^{V_{\mathrm{I}}})^\omega$, there is a unique run σ which is F-compatible and such that $\sigma^{V_{\mathrm{I}}} = \rho$. The *$F$-run tree* is the run tree $t : (S^{V_{\mathrm{I}}})^* \to S^V$ such that each branch is labeled by a word $s_0 s_1 s_2 \cdots \in (S^V)^\omega$ which is an F-run. Note that, in an F-run, the prefix $\sigma[i]$ only depends on the prefix $\rho[i]$. This shows that the F-run tree is unique.

For a variable $v \in V$, we let $\mathrm{View}(v) = (E^{-2})^*(v) \cap V_{\mathrm{I}}$ be the set of input variables v might depend on. Observe that if $s_0\sigma$ is an F-run then, for all $v \in V \setminus V_{\mathrm{I}}$, for all $i > 0$, s_i^v only depends on $\sigma^{\mathrm{View}(v)}[i]$. This allows us to define the summary $\hat{f^v} : (S^{\mathrm{View}(v)})^+ \to S^v$ such that $\hat{f^v}(\sigma^{\mathrm{View}(v)}[i]) = s_i^v$, corresponding to the composition of all local strategies used to obtain the value of v.

Remark 1. The compatibility of the strategies $F = (f^v)_{v \in V \setminus V_{\mathrm{I}}}$ with the delays $(d_v)_{v \in V \setminus V_{\mathrm{I}}}$ extends to the summaries $\hat{F} = (\hat{f^v})_{v \in V \setminus V_{\mathrm{I}}}$. Formally, a map $h : (S^{\mathrm{View}(v)})^+ \to S^v$ is compatible with the delays if for all $\rho \in (S^{\mathrm{View}(v)})^i$, $h(\rho)$ only depends on the prefixes $(\rho^u[i - d(u,v)])_{u \in \mathrm{View}(v)}$, where $d(u,v)$ is the smallest cumulative delay of transmission between u and v, i.e.,

$$d(u,v) = min\{d_{v_1} + \cdots + d_{v_n} \mid u\ E^2\ v_1\ E^2 \ldots E^2\ v_n = v \text{ is a path in } \mathcal{A}\}.$$

The strategy f^v is *memoryless* if it does not depend on the past, that is, if there exists $g : S^{R(v)} \to S^v$ such that $f^v(s_1 \cdots s_i \cdots s_{i+d_v}) = g(s_i)$ for $s_1 \cdots s_{i+d_v} \in (S^{R(v)})^+$. In case $d_v = 0$, this corresponds to the usual definition of a memoryless strategy.

Distributed synthesis problem. Let $\mathcal{L}$ be a specification language. The distributed synthesis problem for an architecture $\mathcal{A}$ is the following: given a formula $\varphi \in \mathcal{L}$, decide whether there exists a distributed program F such that every F-run (or the F-run tree) satisfies φ. We will then say that F is a distributed implementation for the specification φ. If for some architecture the synthesis problem is undecidable, we say that the architecture itself is *undecidable* (for the specification language $\mathcal{L}$).

3 Architectures with Uncomparable Information

In this section, we state a necessary condition for decidability.

Definition 2. *An architecture has* uncomparable information *if there exist variables $x, y \in V_O$ such that $View(x) \setminus View(y) \neq \emptyset$ and $View(y) \setminus View(x) \neq \emptyset$. Otherwise the architecture has* linearly preordered information.

For instance, the architectures of Figures 1 and 3 have linearly preordered information. The following proposition extends the undecidability result of [11,3].

Proposition 3. *Architectures with uncomparable information are undecidable for LTL or CTL external specifications.*

4 Uniformly Well-Connected Architectures

This section introduces the new class of uniformly well-connected (UWC) architectures and provides a decidability criterion for the synthesis problem on this class. It also introduces the notion of *robust* specifications and shows that UWC architectures are always decidable for external and robust specifications.

A *routing* for an architecture $\mathcal{A} = (V \cup P, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ is a family $\Phi = (f^v)_{v\in V\setminus(V_{\mathrm{I}}\cup V_{\mathrm{O}})}$ of *memoryless* local strategies. Observe that a routing does not include local strategies for output variables. Informally, we say that an architecture is uniformly well connected if there exists a routing Φ that allows to transmit to every output variable v, with a minimal delay, the value of the variables in View(v).

Definition 4. *An architecture $\mathcal{A}$ is* uniformly well-connected *(UWC) if there exist a routing Φ and, for every $v \in V_O$ and $u \in$ View(v), a decoding function $g^{u,v} : \left(S^{R(v)}\right)^+ \to S^u$ that can reconstruct the value of u, i.e., such that for any Φ-compatible sequence $\sigma = s_1 s_2 \cdots \in \left(S^{V\setminus V_O}\right)^+$, we have for $i > 0$*

$$s_i^u = g^{u,v}(\sigma^{R(v)}[i + d(u,v) - d_v]) \tag{1}$$

In case there is no delay, the uniform well-connectedness refines the notion of adequate connectivity introduced by Pnueli and Rosner in [11], as we no longer require each output variable to be communicated the value of *all* input variables, but only those in its view. In fact, this gives us strategies for internal variables, that are simply to route the input to the processes writing on output variables.

Observe that, given an architecture, there is a finite number of routings and a finite number of decoding functions, so that the property of being UWC is NP. Actually, the problem is NP-complete: using a natural reduction, this follows from the NP-hardness of the multicast problem [7], which is a special instance of the network information flow problem [1].

We first show that distributed programs are somewhat easier to find in a UWC architecture. As a matter of fact, in such architectures, to define a distributed strategy it suffices to define a collection of input-output strategies that respect the delays given by the architecture.

Lemma 5. *Let $\mathcal{A} = (V \cup P, E, (S^v)_{v\in V}, s_0, (d_p)_{p\in P})$ be a UWC architecture. For each $v \in V_O$, let $h^v : (S^{View(v)})^+ \to S^v$ be an input-output mapping which is compatible with the delays of $\mathcal{A}$. Then there exists a distributed program $F = (f^v)_{v\in V\setminus V_I}$ over $\mathcal{A}$ such that $h^v = \hat{f}^v$ for all $v \in V_O$.*

We now give a decision criterion for this specific subclass of architectures.

Theorem 6. *A UWC architecture is decidable for external (linear or branching) specifications if and only if it has linearly preordered information.*

We have already seen in Section 3 that uncomparable information yields undecidability of the synthesis problem for LTL or CTL external specifications. We prove now that, when restricted to the subclass of UWC architectures, this also becomes a necessary condition.

We assume that the architecture $\mathcal{A}$ is UWC and has linearly preordered information, and therefore we can order the output variables $V_{\mathrm{O}} = \{v_1, \ldots, v_n\}$ so that $\mathrm{View}(v_n) \subseteq \cdots \subseteq \mathrm{View}(v_1) \subseteq V_{\mathrm{I}}$.

In the following, in order to use tree-automata, we extend a strategy $f : (S^X)^+ \to S^Y$ by $f(\varepsilon) = s_0^Y$ so that it becomes a (S^X, S^Y)-tree. We proceed in two steps. First, we build an automaton accepting all the *global input-output 0-delay* strategies implementing the specification. A global input-output 0-delay strategy for $\mathcal{A}$ is a $(S^{\mathrm{View}(v_1)}, S^{V_{\mathrm{O}}})$-tree h satisfying $h(\varepsilon) = s_0^{V_{\mathrm{O}}}$. This first step is simply the program synthesis for a single process with incomplete information (since we may have $\mathrm{View}(v_1) \subsetneq V_{\mathrm{I}}$). This problem was solved in [6] for CTL* specifications.

Proposition 7 ([6, Th. 4.4]). *Given an external specification $\varphi \in CTL^*(V_I \cup V_O)$, one can build a NDTA $\mathfrak{A}_1$ over $(S^{View(v_1)}, S^{V_O})$-trees such that $h \in \mathcal{L}(\mathfrak{A}_1)$ if and only if the run tree induced by h satisfies φ.*

If $\mathcal{L}(\mathfrak{A}_1)$ is empty, then we already know that there are no distributed implementations for the specification φ over $\mathcal{A}$. Otherwise, thanks to Lemma 5, we have to check whether for each $v \in V_{\mathrm{O}}$ there exists an $(S^{\mathrm{View}(v)}, S^v)$-tree h^v which is compatible with the delays and such that the global strategy $\bigoplus_{v \in V_{\mathrm{O}}} h^v$ induced by the collection $(h^v)_{v \in V_{\mathrm{O}}}$ is accepted by $\mathfrak{A}_1$. Formally, the *sum* of strategies is defined as follows. Let $X = X_1 \cup X_2 \subseteq V_{\mathrm{I}}$ and $Y = Y_1 \uplus Y_2 \subseteq V_{\mathrm{O}}$, and for $i = 1, 2$ let h_i be a (S^{X_i}, S^{Y_i})-tree. We define the (S^X, S^Y)-tree $h = h_1 \oplus h_2$ by $h(\sigma) = (h_1(\sigma^{X_1}), h_2(\sigma^{X_2}))$ for $\sigma \in (S^X)^*$.

To check the existence of such trees $(h^v)_{v \in V_{\mathrm{O}}}$, we will inductively eliminate the output variables following the order $v_1, \ldots, v_n$. It is important that we start with the variable that *views* the largest set of input variables, even though, due to the delays, it might get the information much later than the remaining variables. Let $V_k = \{v_k, \ldots, v_n\}$ for $k \geq 1$. The induction step relies on the following statement.

Proposition 8. *Let $1 \leq k < n$. Given a NDTA $\mathfrak{A}_k$ accepting $(S^{View(v_k)}, S^{V_k})$-trees, we can build a NDTA $\mathfrak{A}_{k+1}$ accepting $(S^{View(v_{k+1})}, S^{V_{k+1}})$-trees, such that a tree t is accepted by $\mathfrak{A}_{k+1}$ if and only if there exists a $(S^{View(v_k)}, S^{v_k})$-tree h^{v_k} which is compatible with the delays and such that $h^{v_k} \oplus t$ is accepted by $\mathfrak{A}_k$.*

The proof of Proposition 8 divides in two steps. Since $V_k = \{v_k\} \cup V_{k+1}$, for each $(S^{\mathrm{View}(v_k)}, S^{V_k})$-tree t we have $t = t^{v_k} \oplus t^{V_{k+1}}$ (recall that t^U is the projection of t on U). So one can first turn the automaton $\mathfrak{A}_k$ into $\mathfrak{A}'_k$ that accepts the trees $t \in \mathcal{L}(\mathfrak{A}_k)$ such that t^{v_k} is compatible with the delays (Lemma 9). Then, one can build an automaton that restricts the domain of the directions and the labeling of the accepted trees to $S^{\mathrm{View}(v_{k+1})}$ and $S^{V_{k+1}}$ respectively.

Lemma 9. *Let $v \in U \subseteq V_O$. Given a NDTA $\mathfrak{A}$ over $(S^{View(v)}, S^U)$-trees one can build a NDTA $\mathfrak{A}' = \mathrm{compat}_v(\mathfrak{A})$ also over $(S^{View(v)}, S^U)$-trees such that $\mathcal{L}(\mathfrak{A}') = \{t \in \mathcal{L}(\mathfrak{A}) \mid t^v$ is compatible with the delays$\}$.*

Proof. Intuitively, to make sure that the function t^v is compatible with the delays, the automaton $\mathfrak{A}'$ will guess in advance the values of t^v and then check that its guess is correct. The guess has to be made $K = max\{d(u,v), u \in \text{View}(v)\}$ steps in advance and consists in a function $g : (S^{\text{View}(v)})^K \to S^v$ that is already compatible with the delays and that predicts what will be the v-values K steps later. During a transition, the guess is sent in each direction $r \in S^{\text{View}(v)}$ as a function $r^{-1}g$ defined by $(r^{-1}g)(\sigma) = g(r\sigma)$ which is stored in the state of the automaton. Previous guesses are refined similarly and are also stored in the state of the automaton so that the new set of states is $Q' = Q \times \mathcal{F}$ where $\mathcal{F}$ is the set of functions $f : (S^{\text{View}(v)})^{<K} \to S^v$ which are compatible with the delays, where $Z^{<K} = \bigcup_{i<K} Z^i$. The value $f(\varepsilon)$ is the guess that was made K steps earlier and has to be checked against the current v-value of the tree.

Transitions of $\mathfrak{A}'$ will be defined using the function $\Delta : \mathcal{F} \times S^{\text{View}(v)} \to \mathcal{P}(\mathcal{F})$ given by $\Delta(f,r) = \{f' \mid f'(\sigma) = f(r\sigma) \text{ for } |\sigma| < K-1\}$. Note that the values $f'(\sigma)$ for $|\sigma| = K-1$ do not depend on f and correspond to the new guess g refined by r as intuitively described above. Now, the transition function of $\mathfrak{A}'$ is

$$\delta'\big((q,f),(f(\varepsilon),s)\big) = \left\{ (q_r, f_r)_{r \in S^{\text{View}(v)}} \,\middle|\, \begin{array}{c} (q_r)_{r \in S^{\text{View}(v)}} \in \delta(q,(f(\varepsilon),s)) \text{ and} \\ f_r \in \Delta(f,r) \text{ for all } r \in S^{\text{View}(v)} \end{array} \right\}.$$

Finally, the set of initial states of $\mathfrak{A}'$ is $I' = \{q_0\} \times \mathcal{F}$ and $\alpha' = \pi^{-1}(\alpha)$ where $\pi : (Q \times \mathcal{F})^\omega \to Q^\omega$ is the projection on Q, i.e., a run of $\mathfrak{A}'$ is accepted if and only if its projection on Q is an accepted run of $\mathfrak{A}$. One can check that the automaton $\mathfrak{A}'$ satisfies the requirements of Lemma 9. □

Proof (of Proposition 8). We consider the NDTA $\text{compat}_{v_k}(\mathfrak{A}_k)$. It remains to project away the S^{v_k} component of the label and to make sure that the $S^{V_{k+1}}$ component of the label only depends on the $S^{\text{View}(v_{k+1})}$ component of the input. The first part is the classical projection on $S^{V_{k+1}}$ of the automaton and the second part is the *narrowing* construction introduced in [6]. The automaton $\mathfrak{A}_{k+1}$ fulfilling the requirements of Proposition 8 is therefore given by $\text{narrow}_{\text{View}(v_{k+1})}(\text{proj}_{V_{k+1}}(\text{compat}_{v_k}(\mathfrak{A}_k)))$. Note that, even when applied to a NDTA, the narrowing construction of [6] yields an *alternating* tree automaton. Here we assume that the narrowing operation returns a NDTA using a classical transformation of alternating tree automata into NDTA [9]. The drawback is that this involves an exponential blow up. Unfortunately, this is needed since Lemma 9 requires a NDTA as input. □

We can now conclude the proof of Theorem 6. Using Proposition 8 inductively starting from the NDTA $\mathfrak{A}_1$ of Proposition 7, we obtain a NDTA $\mathfrak{A}_n$ accepting a $(S^{\text{View}(v_n)}, S^{v_n})$-tree h^{v_n} if and only if for each $1 \le i < n$, there exists a $(S^{\text{View}(v_i)}, S^{v_i})$-tree h^{v_i} which is compatible with the delays and such that $h^{v_1} \oplus \cdots \oplus h^{v_n}$ is accepted by $\mathfrak{A}_1$. Therefore, using Remark 1 and Lemma 5, there is a distributed implementation for the specification over $\mathcal{A}$ if and only if $\mathcal{L}(\text{compat}_{v_n}(\mathfrak{A}_n))$ is nonempty. The overall procedure is non-elementary due to the exponential blow-up of the inductive step in Proposition 8. □

We now show that we can obtain decidability of the synthesis problem for the whole subclass of UWC architectures by restricting ourselves to specifications that only relate output variables to their own view.

Definition 10. *A specification $\varphi \in \mathcal{L}$ with $\mathcal{L} \in \{LTL, CTL, CTL^*\}$ is* robust *if it is a (finite) disjunction of formulas of the form $\bigwedge_{v \in V_O} \varphi_v$ where $\varphi_v \in \mathcal{L}(View(v) \cup \{v\})$.*

Proposition 11. *The synthesis problem for UWC architectures and external robust CTL* specifications is decidable.*

Proof. Let $\mathcal{A} = (V \cup P, E, (S^u)_{u \in V}, s_0, (d_p)_{p \in P})$ be a UWC architecture and φ be an external and robust CTL* specification. Without loss of generality, we may assume that $\varphi = \bigwedge_{v \in V_\mathrm{O}} \varphi_v$ where $\varphi_v \in \mathrm{CTL}^*(\mathrm{View}(v) \cup \{v\})$. Using Proposition 7, for each $v \in V_\mathrm{O}$ we find a NDTA $\mathfrak{A}_v$ accepting a strategy $h : (S^{\mathrm{View}(v)})^* \to S^v$ if and only if the induced run tree $t_v : (S^{\mathrm{View}(v)})^* \to S^{\mathrm{View}(v) \cup \{v\}}$ satisfies φ_v. Using Remark 1 and Lemma 5 one can show the following claim from which Proposition 11 follows.

Claim. There exists a distributed implementation of φ over $\mathcal{A}$ if and only if for each $v \in V_\mathrm{O}$, the automaton $\mathrm{compat}_v(\mathfrak{A}_v)$ is nonempty. □

5 Well-Connected Architectures

It is natural to ask whether the decision criterion for UWC architectures can be extended to a larger class. In this section, we relax the property of uniform well-connectedness and show that, in that case, linearly preordered information is not anymore a sufficient condition for decidability.

Definition 12. *An architecture is said to be* well-connected*, if for each output variable $v \in V_O$, the sub-architecture consisting of $(E^{-1})^*(v)$ is uniformly well-connected.*

The architecture of Figure 2 is well-connected but not UWC when the variables are boolean. This follows from similar results on the multicast problem [7]. Hence, the subclass of UWC architectures is strictly contained in the subclass of well-connected architecture. Note that the size of the variable domains has a major influence: any well-connected architecture with sufficiently large domain sizes is UWC.

The following theorem asserts that, unfortunately, the decision criterion cannot be extended to well-connected architectures.

Theorem 13. *The synthesis problem for LTL specifications and well-connected, linearly preordered architectures is undecidable.*

Let $\mathcal{A}$ be the architecture of Figure 3, in which all the delays are set to 0, and which is clearly well-connected and linearly preordered. To show its undecidability, fix a deterministic Turing machine M with tape alphabet Γ and state set Q.

We reduce the non halting problem of M starting from the empty tape to the distributed implementability of an LTL specification over $\mathcal{A}$. Let $S^z = \{0,1\}$ for $z \in V \setminus \{x,y\}$ and $S^x = S^y = \Gamma \uplus Q \uplus \{\#\}$ where # is a new symbol. As usual, the configuration of M defined by state q and tape content $\gamma_1\gamma_2$, where the head scans the first symbol of γ_2, is encoded by the word $\gamma_1 q \gamma_2 \in \Gamma^* Q \Gamma^+$. An input word $u \in 0^* 1^p 0\{0,1\}^\omega$ encodes the integer $n(u) = p$ and similarly for v. We construct an LTL specification φ_M forcing any distributed implementation to output on variable x the $n(u)^{\text{th}}$ configuration of M starting from the empty tape. Processes p_0 and p_6 play the role of the two processes of the undecidable architecture of Pnueli and Rosner. The difficulty is to ensure that process p_6 cannot receive relevant information about u.

The specification φ_M is a conjunction of five properties described below that can all be expressed in $\text{LTL}(V_\text{I} \cup V_\text{O})$.

1. The processes p_i for $i \neq 6$ have to output the current values of u and w until (including) the first 1 occurs on w. Afterwards, they are unconstrained. Process p_6 must always output the value of w on w_6. Moreover, after the first 1 on w, it also has to output the current value of u on u_6. We can describe this property with a formula α.
2. If the input word on u (resp. v) is in $0^q 1^p 0\{0,1\}^\omega$, then the corresponding output word x (resp. y) is in $\#^{q+p}\Gamma^* Q \Gamma^+ \#^\omega$. This can be expressed by a formula β.
3. We next express with a formula γ that if $n(u) = 1$, then the output on x is the first configuration $\mathcal{C}_1$ of M starting from the empty tape.
4. We say that the input words are *synchronized* if $u, v \in 0^q 1^p 0\{0,1\}^\omega$ or if $u \in 0^q 1^{p+1} 0\{0,1\}^\omega$ and $v \in 0^{q+1} 1^p 0\{0,1\}^\omega$. We use a formula δ to express the fact that if u and v are synchronized and $n(u) = n(v)$, then the outputs on x and y are equal.
5. Finally, one can express with an LTL formula ψ that if the input words are synchronized and if $n(u) = n(v) + 1$, then the configuration encoded on x is obtained by a computation step of M from the configuration encoded on y.

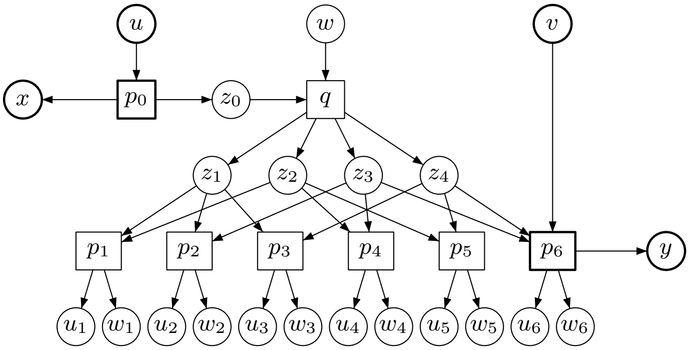


Fig. 3. Undecidable, well-connected, comparable-information architecture

We first show that there exists a distributed implementation of φ_M over $\mathcal{A}$. Let $\oplus$ be the addition modulo 2 (XOR). Process p_0 forwards u to z_0. Process q forwards u to z_1, $u \oplus w$ to z_2 and w to z_3. The strategy for z_4 is not memoryless. Process q forwards w to z_4 until (including) the first 1 on w and then it forwards $u \oplus w$ to z_4. Formally, $f^{z_4}(u, 0^q b) = b$ and $f^{z_4}(ub_1, 0^q 1 w b_2) = b_1 \oplus b_2$. We also use memoryless strategies for the processes p_i so that α is satisfied. For instance, the strategy for p_1 is $f^1(b_1, b_2) = (b_1, b_1 \oplus b_2)$ and the strategy for p_6 (y excluded) is $f^6(b_3, b_4) = (b_3 \oplus b_4, b_3)$. It is easy to see that with these strategies, the first property α of the specification is satisfied.

The strategy f^x (respectively f^y) is to output the p^{th} configuration of M starting from the empty tape when u (respectively v) encodes p. Then, the rest of the specification, $\beta \wedge \gamma \wedge \delta \wedge \psi$, is satisfied.

Remark 14. Actually, one can define another distributed implementation by changing only the strategy f^{z_4}: at each step, process q transmits to p_6 the value of u *at the preceding step* as the mod 2 difference between z_3 and z_4, until the first 1 occurs on w. Formally, $f^{z_4}(u \cdot a_1 \cdot a_2, 0^q b) = a_1 \oplus b$ and we adapt the strategies of $p_1, \ldots, p_6$ so that α is satisfied. By XORing its two arguments, process p_6 can then recover the whole history of u, except the bit occurring simultaneously with the first 1 of w. Hence, we are almost in the situation of the decidable architecture of Figure 1, but surprisingly, *missing only one bit of information* suffices to induce undecidability.

Let now $F = (f^v)_{v \in V \setminus V_I}$ be a distributed implementation of φ_M on the architecture $\mathcal{A}$ of Figure 3. We prove that f^x must simulate the computation of M starting from the empty tape.

Let $q \geq 0$. For $u = 0^q 1 u'$, we define $u^0 = 0^q 0 u'$. The next lemma states that strategies f^{z_3} (resp. f^{z_4}) must output the same sequence for u and u^0 if the input word w is suitable. This is the main technical lemma whose proof relies on the specification α.

Lemma 15. *Let $u, w \in 0^q 1 \{0,1\}^\omega$. For $k \in \{3, 4\}$, we have for all $n > 0$:*

$$\hat{f}^{z_k}(u^0[n], w[n]) = \hat{f}^{z_k}(u[n], w[n]). \tag{2}$$

Lemma 16. *If x is computed by f^x from the input word u then for all $p > 0$ we have*

$$\forall q \geq 0, \qquad u \in 0^q 1^p 0 \{0,1\}^\omega \Longrightarrow x = \#^{p+q} \mathcal{C}_p \#^\omega \tag{3}$$

where $\mathcal{C}_p$ is the p-th configuration reached by M starting from the empty tape.

Proof. The proof is by induction on p. The case $p = 1$ follows from the specification γ. Assume now that $u \in 0^q 1^{p+1} 0 \{0,1\}^\omega$ and let $v = 0^{q+1} 1^p 0^\omega$ and $w = 0^q 1^\omega$. By induction, for $u^0 \in 0^{q+1} 1^p 0 \{0,1\}^\omega$ the output is $x = \#^{q+1+p} \mathcal{C}_p \#^\omega$. Using δ, we deduce that on the input triple (u^0, v, w) the output is $y = x = \#^{q+1+p} \mathcal{C}_p \#^\omega$. Now, by Lemma 15, on the input pairs (u^0, w) and (u, w), the outputs on z_3 and z_4 are the same. Hence, on the input triples (u^0, v, w) and (u, v, w) the outputs on y must be $y = \#^{q+1+p} \mathcal{C}_p \#^\omega$ by the above. Using ψ, we deduce that on the input triple (u, v, w) the output on x must be $x = \#^{q+1+p}\mathcal{C}_{p+1}\#^{\omega}$. This concludes the proof since x only depends on u. □

It is then easy to get the undecidability of the architecture $\mathcal{A}$ of Figure 3 by considering the specification $\varphi_M \wedge \mathsf{G}(x \neq \text{halt})$.

6 Conclusion

In this paper, we have argued that it is meaningful to rule out specifications for distributed architectures constraining internal variables. We have shown that every decidable architecture is linearly preordered, and that this condition is sufficient for deciding external specifications on UWC architectures. On the other hand, we have exhibited a linearly preordered, yet undecidable well-connected architecture for external LTL specifications, by simulating the loss of a single information bit on the UWC architecture of Figure 1.

Finally, we have shown that all UWC architectures are decidable for *external* and *robust* specifications, i.e., specifications constraining external variables which are causally related by a communication path. A challenging problem is to find whether this still holds for well-connected architectures.

References

1. R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. Inform. Theory*, 46(4):1204–1216, 2000.
2. A. Church. Logic, arithmetic, and automata. In *Int. Symp. of Mathematicians*, pages 23–35, 1962.
3. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. 20th IEEE Symp. on Logic in Computer Science (LICS 2005)*. IEEE Computer Society, 2005.
4. P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. Technical report, LSV, 2006.
5. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proceedings of LICS'01*. Computer Society Press, 2001.
6. O. Kupferman and M. Y. Vardi. Church's problem revisited. *The Bulletin of Symbolic Logic*, 5(2):245–263, June 1999.
7. A. R. Lehman and E. Lehman. Complexity classification of network information flow problems. In *Proceedings of SODA'04*, pages 142–150. SIAM, 2004.
8. P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *Proceedings of ICALP'01*, volume 2076 of *Lect. Notes Comp. Sci.*, pages 396–407. Springer, 2001.
9. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of theorems of Rabin, McNaughton and Safra. *Theoret. Comput. Sci.*, 2(1):90–121, 1995.
10. G. Peterson and J. Reif. Multiple-person alternation. In *20th Annual Symposium on Foundations of Computer Science (San Juan, Puerto Rico, 1979)*, pages 348–363. IEEE, New York, 1979.
11. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings of 31th IEEE Symp. FOCS*, pages 746–757, 1990.

The Meaning of Ordered SOS

MohammadReza Mousavi[1,2,⋆], Iain Phillips[3],
Michel A. Reniers[1], and Irek Ulidowski[4]

[1] Eindhoven University of Technology, Eindhoven, The Netherlands
[2] Reykjavík University, Reykjavík, Iceland
[3] Imperial College, London, United Kingdom
[4] University of Leicester, Leicester, United Kingdom

Abstract. Structured Operational Semantics (SOS) is a popular method for defining semantics by means of deduction rules. An important feature of deduction rules, or simply SOS rules, are *negative premises*, which are crucial in the definitions of such phenomena as priority mechanisms and time-outs. Orderings on SOS rules were proposed by Phillips and Ulidowski as an alternative to negative premises. The meaning of general types of SOS rules with orderings has not been studied hitherto. This paper presents satisfactory ways of giving a meaning to general SOS rules with orderings. We also give semantics-preserving transformations between the two paradigms, namely, SOS with negative premises and SOS with orderings.

1 Introduction

It is well-known that negative premises in Structured Operational Semantics (SOS) are useful and non-trivial additions but at the same time they may lead to ambiguities and paradoxical phenomena with respect to the semantics of SOS [4,5]. As an alternative to negative premises, [9] proposes to furnish SOS deduction rules with an ordering. But to avoid the same difficulties as those with negative premises, [9] restricts itself to the positive subset of GSOS [3] which does not allow for look-ahead or complex terms as sources of the premises.

It is also well-known from the term rewriting literature that the introduction of orderings (called priorities) to term rewrite systems introduces challenges for the well-definedness of the semantics of term rewrite systems [2]. SOS specifications can be seen as conditional term rewrite systems and thus one expects similar or even more difficult challenges when studying the general semantics of SOS with ordering.

A fundamental study of the semantics of ordered SOS (in its full generality) has not been carried out to date and even misconceptions exist. In [8, Theorem 4], it is mentioned (without formal proof) that they can generalize their particular rule format for ordered SOS with look-ahead while preserving the congruence

⋆ The work of the first author has been partially supported by the project "The Equational Logic of Parallel Processes" (nr. 060013021) of The Icelandic Research Fund.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 333–344, 2006.

property. However, as we show in the remainder of this paper, the introduction of either look-ahead or complex terms as sources of premises to ordered SOS jeopardizes the well-definedness of the induced transition relation (let alone the congruence result).

In the remainder of this paper, in Section 2, we define the basic concept of Ordered Transition System Specification (OTSS) which is a general framework for ordered SOS. In the same section, we give some examples, both for illustrating the applications of ordered SOS and for showing that the semantics of OTSS's is not always clear. Then, in Section 3, following [5], we define a model-theoretic and a proof-theoretic view to the meaning of ordered SOS and prove them equal. Subsequently, in Sections 4 and 5, we give semantics-preserving transformations from a novel rule format for ordered SOS (called otyft, for order tyft, where tyft is a coding for the terms admitted in the deduction rules) to a rule format for SOS with negative premises (called ntyft [6]) and vice versa. In Section 6 we show that our otyft rule format indeed induces congruence for strong bisimilarity. Section 7 concludes the paper.

2 (Ordered) Transition System Specification

2.1 Basic Concepts

Definition 1 (Signature, Term and Substitution). Assume a countable set of variables V (with typical members x, y, x', y', x_i, $y_i \ldots$). A *signature* Σ is a set of function symbols (operators, with typical members f, g, $\ldots$) with fixed arities $ar : \Sigma \to I\!N$. Functions with zero arity are called constants and are typically denoted by a, b, c and d. Terms $s, t, t_i, \ldots \in \mathcal{T}$ are constructed inductively using variables and function symbols. A list of terms is denoted by $\overrightarrow{t}$. When we write $f(\overrightarrow{t})$, we assume that $\overrightarrow{t}$ has the right size, i.e., $ar(f)$. All terms are considered open terms. Closed terms $p, q, \ldots \in \mathcal{C}$ are terms that do not contain a variable and are typically denoted by $p, q, l, p', p_i, \ldots$. The set of variables appearing in term t is denoted by $vars(t)$.

Definition 2 (Ordered Transition System Specification (OTSS)). Given a signature and a set of variables, a *Transition System Specification (TSS)* is a set R of deduction rules.

A deduction rule $r \in R$, is defined as a tuple (H, c) where H is a set of formulae and c is a positive formula. For all $t, t' \in \mathcal{T}$ and $l \in \mathcal{C}$ we define that $\phi = t \xrightarrow{l} t'$ is a positive formula and $\phi' = t \overset{l}{\nrightarrow}$ is a negative formula. A formula is a positive or a negative formula. We denote the set of formulae by Φ and the set of positive fomulae by Φ_p. Term t is called the source of both ϕ and ϕ', denoted by $src(\phi)$ and $src(\phi')$, and t' is called the target of ϕ. The formula c is called the *conclusion* of r, denoted by $conc(r)$, and the formulae in H are called its *premises* and denoted by $prem(r)$. A positive deduction rule (TSS) is a deduction rule of which all the premises (all the deduction rules) are positive. The notions of source and target generalize to a set of formulae, as expected.

Also, the notion of "variables of" is naturally lifted to sets of terms, formulae, sets of formulae and deduction rules.

An *Ordered Transition System Specification (OTSS)* is a pair $(R, <)$ where R is a *positive TSS* and $< \subseteq R \times R$ is an *arbitrary relation* on the deduction rules. For a rule r, $\mathit{higher}(r)$ is defined as $\{r' \mid r' \in R \wedge r' > r\}$, i.e., the set of rules placed above r by the ordering $<$.

The intuition behind the ordering on rules is that a rule r can only be applied when all rules $r' \in \mathit{higher}(r)$ are disabled since they do not have a "reason" (or "proof") for their premises to hold. As we show in the remainder, this notion of "reason" or "proof" is not trivial to define and involves the same complications as those concerning the semantics of TSS's with negative premises [5].

2.2 Rule Formats

A major line of research in the SOS meta-theory concerns defining syntactic schema for TSS's which guarantee certain properties such as congruence of strong bisimilarity. A distinguished example of such rule formats is the ntyft rule format due to [6] which has powerful and complicating features such as look-ahead and negative premises.

Definition 3 ((N)Tyft). A rule is in the ntyft rule format when it is of the form $\dfrac{\{t_i \xrightarrow{l_i} y_i \mid i \in I\} \cup \{t_j \overset{l_j}{\nrightarrow} \mid j \in J\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$ where all variables in $\overrightarrow{x}$ and y_i's are pairwise distinct (i.e., for all $i, i' \in I$ and $1 \leq j < j' \leq ar(f)$, $y_i \neq x_j$, $x_j \neq x_{j'}$ and if $i \neq i'$ then $y_i \neq y_{i'}$), f is a function symbol from the signature, I and J are (possibly infinite) sets of indices, t, t_i's and t_j's are arbitrary terms and l, l_i's and l_j's are closed terms.

A TSS is in the ntyft rule format when all its rules are. A rule (TSS) is in the tyft rule format when it is positive and in the ntyft rule format.

Our goal is to show that ordering on rules is at least as expressive (and of course complicated in nature) as negative premises and thus, we introduce the following otyft rule format which will be proved equal in expressiveness to the ntyft rule format (in Sections 4 and 5).

Definition 4 (Otyft). An OTSS $(R, <)$ is in the otyft rule format when (1) for all rules $r \in R$, either r is in the tyft rule format or $\mathit{conc}(r) \in \mathit{prem}(r)$, (2) for all rules $r, r' \in R$ such that $r' \in \mathit{higher}(r)$ (a) if a premise of r has the same target as that of a premise of r', then the two premises are the same (i.e., have the same source and label) and (b) $\mathit{vars}(\mathit{src}(\mathit{prem}(r'))) \subseteq (\mathit{vars}(\mathit{prem}(r)) \cup \mathit{vars}(\mathit{src}(\mathit{conc}(r))))$.

The above rule format generalizes the OSOS rule format of [9] by allowing for look-ahead and arbitrary terms as sources of premises (both conditions 2.a and

2.b are trivially satisfied by OTSS's in the OSOS rule format). In the forthcoming extended version of this paper, we prove that removing condition 2.b gives rise to a more general rule format called universal otyft, while preserving the congruence result. The expressiveness of this rule format, called universal otyft, goes beyond that of the ntyft rule format in that all transition relations specifiable by a TSS in the ntyft rule format can be specified by an OTSS in the universal otyft rule format but *not* vice versa.

The non-tyft rules allowed by the otyft rule format are mainly for convenience: our definitions of semantics in Section 3 are insensitive to such rules and they are useful in the translation between the ntyft and the otyft rule formats in Section 4.

2.3 Examples

Orderings on positive rules can replace negative premises in rules [9]. In the remainder, we start with a simple example motivating the use of ordering (as an alternative to negative premises). Then we show that our new otyft rule format extends the applicability of the ordered SOS paradigm by specifying an example involving look-ahead. Finally, we show that this extension comes at a price, namely, the semantics of general OTSS's (e.g., those involving look-ahead) is not always clear and should be studied more thoroughly.

Example 5 (Priority). The priority operator θ [1] may be used to represent such phenomena as time-outs and interrupts. For a given partial order $\prec$ on actions (a set of constants, denoted by a, b, c, $\ldots \in Act$), $\theta(p)$ is a restriction on the behavior of p such that action a can happen only if no b with $a \prec b$ is possible. If $B_a = \{b \mid a \prec b\}$, then θ can be defined by this TSS (where the rule is actually a rule schema which should be repeated for each action $a \in Act$): $\dfrac{x \xrightarrow{a} y \quad \{x \overset{b}{\nrightarrow} \mid b \in B_a\}}{\theta(x) \xrightarrow{a} \theta(y)}$.

Alternatively, θ can be defined by positive rules r_a, equipped with the ordering defined by $r_a < r_b$ whenever $a \prec b$: $(r_a)\dfrac{x \xrightarrow{a} y_a}{\theta(x) \xrightarrow{a} \theta(y_a)}$ where y_a are distinct variables for all $a \in Act$. (Note that the naming of variables in the rules related by ordering is indeed important; if for two different actions a and b such that $a \prec b$, $y_a = y_b$, then the OTSS specified by the above rules is not in the otyft rule format and as shown in Section 6, it may lack intuitive properties such as congruence of bisimilarity.)

Example 6 (Timed Parallel Composition). Consider the following TSS defining the semantics of a subset of Hennessy and Regan's Process Algebra for Timed Systems (TPA) [7]. The signature consists of a constant *nil*, unary operators $a.\_$ (action prefixing, for all $a \in Act$), $\tau.\_$ (internal action prefixing) and$\sigma._$ (time step prefixing) and a binary operator $\|$ (parallel composition). (Constants a, τ σ are also introduced in the signature to model the labels.)

$$(a)\frac{}{a.x \xrightarrow{a} x} \quad (\tau)\frac{}{\tau.x \xrightarrow{\tau} x} \quad (\sigma_0)\frac{}{\sigma.x \xrightarrow{\sigma} x} \quad (\sigma_1)\frac{}{a.x \xrightarrow{\sigma} a.x} \quad (\sigma_2)\frac{}{nil \xrightarrow{\sigma} nil}$$

$$(\|_0)\frac{x_0 \xrightarrow{a} y_0}{x_0 \parallel x_1 \xrightarrow{a} y_0 \parallel x_1} \quad (\|_1)\frac{x_1 \xrightarrow{a} y_1}{x_0 \parallel x_1 \xrightarrow{a} x_0 \parallel y_1}$$

$$(\tau_0)\frac{x_0 \xrightarrow{\tau} y_0}{x_0 \parallel x_1 \xrightarrow{\tau} y_0 \parallel x_1} \quad (\tau_1)\frac{x_1 \xrightarrow{\tau} y_1}{x_0 \parallel x_1 \xrightarrow{\tau} x_0 \parallel y_1}$$

$$\textbf{(comm)}\frac{x_0 \xrightarrow{a} y_0 \quad x_1 \xrightarrow{\overline{a}} y_1}{x_0 \parallel x_1 \xrightarrow{\tau} y_0 \parallel y_1} \quad \textbf{(time)}\frac{x_0 \xrightarrow{\sigma} y_0 \quad x_1 \xrightarrow{\sigma} y_1 \quad x_0 \parallel x_1 \overset{\tau}{\nrightarrow}}{x_0 \,||\, x_1 \xrightarrow{\sigma} y_0 \,||\, y_1}$$

In the semantics of the parallel composition operator, $p \parallel q$ can pass time (denoted by label σ) if both p and q can pass time, and if they are stable and cannot communicate (i.e. $p \parallel q \overset{\tau}{\nrightarrow}$).

The above semantics can be specified in ordered SOS by placing a positive version of the rule **(time)** below the rules (τ_0), (τ_1) and **(comm)** as shown below. All other rules are copied to the following OTSS and are unrelated (in terms of ordering) to the rules below.

↓	$(\tau_0)\frac{x_0 \xrightarrow{\tau} y_0}{x_0 \parallel x_1 \xrightarrow{\tau} y_0 \parallel x_1}$	$(\tau_1)\frac{x_1 \xrightarrow{\tau} y_1}{x_0 \parallel x_1 \xrightarrow{\tau} x_0 \parallel y_1}$	$\textbf{(comm)}\frac{x_0 \xrightarrow{a} y_0 \quad x_1 \xrightarrow{\overline{a}} y_1}{x_0 \parallel x_1 \xrightarrow{\tau} y_0 \parallel y_1}$
	$\textbf{(time)}\frac{x_0 \xrightarrow{\sigma} y_0' \quad x_1 \xrightarrow{\sigma} y_1'}{x_0 \parallel x_1 \xrightarrow{\sigma} y_0' \parallel y_1'}$		

We fix the above notation for ordering so that in each column, rules of the upper row have priority over rules of the lower row, i.e., rules of the lower row can only be "applied" when no rule in the upper row (of the same column) can be "applied". Formally, we have the following orderings: (τ_0) > **(time)**, (τ_1) > **(time)**, and **(comm)** > **(time)**.

In the following example, we address the idea of extending OSOS [9] with look-ahead as suggested by [8, Theorem 4] and show that it may lead to pathological specifications with an unclear meaning. (The rule format of [8] extends traditional OTSS with probabilities but the problem we address below is orthogonal to the presence or absence of probabilities and hence, we use the plain OTSS setting as defined before.)

Example 7 (OSOS with Look-Ahead). Consider the OTSS with the following rules. Note that according to the notation fixed before, in the following OTSS, it holds that $\frac{x \xrightarrow{b} y \quad y \xrightarrow{d} z}{f(x) \xrightarrow{d} d} > \frac{x \xrightarrow{b} y}{f(x) \xrightarrow{c} c}$ and $\frac{x \xrightarrow{a} y \quad y \xrightarrow{c} z}{g(x) \xrightarrow{c} c} > \frac{x \xrightarrow{a} y}{g(x) \xrightarrow{d} d}$ but it does not hold that $\frac{}{a \xrightarrow{a} f(a)} > \frac{x \xrightarrow{b} y}{f(x) \xrightarrow{c} c}$.

↓	$\frac{x \xrightarrow{b} y \quad y \xrightarrow{d} z}{f(x) \xrightarrow{d} d}$	$\frac{x \xrightarrow{a} y \quad y \xrightarrow{c} z}{g(x) \xrightarrow{c} c}$	$\frac{}{a \xrightarrow{a} f(a)}$	$\frac{}{a \xrightarrow{b} g(a)}$
	$\frac{x \xrightarrow{b} y}{f(x) \xrightarrow{c} c}$	$\frac{x \xrightarrow{a} y}{g(x) \xrightarrow{d} d}$		

At first sight, it is not intuitively clear which of the following three transition relations should be considered as the meaning of the above OTSS.

1. $\{a \xrightarrow{a} f(a), a \xrightarrow{b} g(a), f(a) \xrightarrow{c} c, g(a) \xrightarrow{c} c\}$ or
2. $\{a \xrightarrow{a} f(a), a \xrightarrow{b} g(a), f(a) \xrightarrow{d} d, g(a) \xrightarrow{d} d\}$ or
3. $\{a \xrightarrow{a} f(a), a \xrightarrow{b} g(a)\}$.

So, a convincing semantics for OTSS's should either be neutral about different possibly derivable transitions (in items 1 and 2) or reject the above OTSS altogether due to its ambiguous nature. We present solutions that cater for both possibilities in the remainder of this paper.

The situation with the following OTSS is even worse.

↓	$\frac{x \xrightarrow{a} y \quad y \xrightarrow{b} z}{f(x) \xrightarrow{b} a}$	$\frac{}{a \xrightarrow{a} f(a)}$
	$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{b} b}$	

If one initially assumes that from rules in the first row one cannot derive any transition with $f(a)$ as its source (which is a legitimate assumption), then the rule below allows for deriving $f(a) \xrightarrow{b} b$. This transition, in turn, enables the premises of the rule above it (leading to the conclusion that $f(a) \xrightarrow{b} a$ should be derivable) and thus the very same rule below must have been disabled and the chain of contradictory conclusions goes forever. Again, any convincing semantics for OTSS's should either find a way to deal with the contradicting conclusions (e.g., by considering all of them uncertain, yet possibly, derivable transitions) or reject the above OTSS altogether due to its paradoxical nature. The notions of semantics presented in the remainder allow for both interpretations.

The above examples make the case for a more profound study of the meaning of ordered SOS which is the subject of the following section.

3 Semantics of OTSS

An OTSS is supposed to induce a unique transition relation on closed terms but as Example 7 already suggested, for some OTSS's the way to assign such a transition relation may not be straightforward. This phenomenon has been known in several areas such as logic programming and term rewriting and even inside the SOS meta-theory as the result of introducing negative premises to SOS rules. For TSS's with negative premises, several notions of semantics have been defined and used of which [5] provides an overview and a comparison. In this paper, due to lack of space, we only present two very general model-theoretic and proof-theoretic approaches to giving semantics to OTSS's. To avoid repeating the phrase "an instance of rule r under a closing substitution σ", in this section, we assume that the OTSS's only contain closed terms. To define the semantics of an arbitrary OTSS, one may instantiate the rules and the ordering relation under all closing substitutions and then use the notions of the semantics in the remainder of this section.

We start with the following notion of provability which is the usual way of giving semantics to ordinary positive TSS's (i.e., without ordering or negative premises).

Definition 8 (Proof). Given an OTSS $(R, <)$, a proof p for a formula ϕ is a well-founded upwardly branching tree of which

1. the nodes are formulae,
2. the root is ϕ, and
3. if a node is labelled ϕ' and the nodes above it form the set K; then there is a rule $r = \frac{K}{\phi'} \in R$.

An r-proof for ϕ is a proof in which the last step is due to rule r. We write $\vdash_p \phi$ when p is a proof in $(R, <)$ for ϕ. We denote the set of rules used in a proof p by $rules(p)$.

3.1 Model-Theoretic Solution

For OTSS's the above notion of provability is too lax because it neglects the ordering among rules. Hence, we have to provide an addendum to the above concept which makes sure that the rules placed above those used in the proof are disabled. The first way to specify this addendum is the following (model-theoretic) notion of correctness.

Definition 9 (Correct). Given an OTSS $(R, <)$ and a transition relation T, we say that a rule $r = \frac{H}{\phi} \in R$ is correct w.r.t. T when for all $r' = \frac{H'}{\psi} \in higher(r)$, $H' \nsubseteq T$.

Our first solution is based on the following notion of three-valued stable model. Such three-valued solutions assign three transition relations to each OTSS, namely the set of transitions that are certainly derivable denoted by C, transitions that are possibly derivable denoted by P (thus $C \subseteq P$) and the set of transitions that are impossible denoted by I. Three-valued solutions may be written as pairs of these sets, i.e., (C, P) or (C, I), with the third component being easily constructed given the other two. Later in this section, we discuss how to adopt the three-valued stable model to define a single transition relation for an OTSS.

Definition 10 (Three-Valued Stable Model). Given an OTSS $(R, <)$, a pair of transition relations (C, P) is a three-valued stable model when $C \subseteq P$ and

1. $\phi \in C \Leftrightarrow \vdash_p \phi$ for some proof p such that $\forall_{r \in rules(p)}$ r is correct w.r.t. P and
2. $\phi \in P \Leftrightarrow \vdash_p \phi$ for some proof p such that $\forall_{r \in rules(p)}$ r is correct w.r.t. C.

The third value of the stable model I, for impossible, is the set of transitions that are not included in P. Of particular interest, among three-valued stable models, is the least one with respect to the information ordering, i.e., $(C, P) < (C', P')$ when $C \subseteq C'$ and $P' \subseteq P$.

The following reduction technique [4] is a method to calculate the least three-valued stable model (thus such a least model indeed exists).

Definition 11 (Reduction Technique). For an ordinal α, define:

$$\begin{aligned}
C_0 &\doteq \emptyset \\
U_0 &\doteq \Phi_p \\
C_\alpha &\doteq \{\phi \mid \ \vdash_p \phi \wedge \exists_{\beta<\alpha} \forall_{r \in rules(p)} r \text{ is correct w.r.t. } C_\beta \cup U_\beta\} \\
U_\alpha &\doteq \{\phi \mid \ \vdash_p \phi \wedge \forall_{\beta<\alpha} \forall_{r \in rules(p)} r \text{ is correct w.r.t. } C_\beta\}
\end{aligned}$$

Lemma 12. Given an OTSS $(R, <)$, for all ordinals α and β such that $\alpha < \beta$, the following statements hold:

1. $C_\alpha \subseteq C_\beta$;
2. $U_\beta \subseteq U_\alpha$;
3. $C_\beta \subseteq C_\alpha \cup U_\alpha$;
4. $C_\beta \cup U_\beta \subseteq C_\alpha \cup U_\alpha$.

From items 1 and 2 of the above lemma (and Tarski's fixpoint theorem), it follows that both C_α and U_α will reach fixpoints, which we denote by C and U, respectively. From item 1-4 and Definition 11, it follows that $(C, C \cup U)$ is the least three-valued stable model of the OTSS under consideration.

Example 13. Consider the first OTSS of Example 7. Its three-valued stable model consists of a certain component $C = \{a \xrightarrow{a} f(a), a \xrightarrow{b} g(a)\}$ and a possible component $P = \{a \xrightarrow{a} f(a), a \xrightarrow{b} g(a),\ f(a) \xrightarrow{c} c, g(a) \xrightarrow{c} c,\ f(a) \xrightarrow{d} d, g(a) \xrightarrow{d} d\}$.

Similarly, for the second OTSS of Example 7, the certain component of the least three-valued stable model only contains $a \xrightarrow{a} f(a)$ and the possible component contains $a \xrightarrow{a} f(a)$ as well as both $f(a) \xrightarrow{b} a$ and $f(a) \xrightarrow{b} b$.

Now the question is how to reduce the three-valued model to a two-valued one, i.e., to associate a unique transition relation to a (meaningful) OTSS. The following notions provide two plausible answers to this question.

Semantics 1 (Complete). An OTSS is meaningful when for its least three-valued stable model (C, P) it holds that $C = P$ (such an OTSS is called *complete*) and its meaning is the least three-valued stable model.

In order to obtain useful meta-results, e.g., the congruence meta-result (discussed in Section 6), one has to restrict attention to complete OTSS's and for practical applications the OTSS under consideration should be complete or should be rejected. However, one might want to generalize Semantics 1 to the following notion of irrefutability which assigns a transition relation to all OTSS's.

Semantics 2 (Irrefutable). All OTSS are meaningful and their meaning is the P component of their least three-valued stable model.

3.2 Proof-Theoretic Solution

An alternative way of giving semantics to OTSS's is by means of a well-supported proof. In a well-supported proof, in addition to constructing a traditional proof, we provide a "proof" for inapplicability of the higher rules; such a "proof" is called a *well-supported denial.* A well-supported denial makes sure that a formula is not derivable since all proofs leading to the formula contain a rule that is provably disabled (i.e., there is a higher rule that has a well-supported proof for all of its premises).

Definition 14 (Well-Supported Proof). Given an OTSS $(R, <)$ and a rule $r \in R$, a well-supported r-proof (or just a well-supported proof) for ϕ is a well-founded upwardly branching tree of which

1. the nodes are formulae,
2. the root is ϕ,
3. if a node is labelled ϕ' and the nodes above it form the set K, then there is a rule $r' = \frac{K'}{\phi'} \in R$ such that $K' \subseteq K$ (for the root node, $r' = r$) and for all $r'' = \frac{H'}{\psi} \in \mathit{higher}(r')$, there exists a set $D_{\psi'} \subseteq K$ denying some $\psi' \in H'$ by a well-supported proof.

A set D_ϕ denies a formula ϕ when for all proofs p such that $\vdash_p \phi$ (in the sense of Definition 8), there exists a rule $r \in \mathit{rules}(p)$ and there exists a rule $r' = \frac{H'}{\phi'} \in \mathit{higher}(r)$ such that $H' \subseteq D_\phi$. The structure providing a well-supported proof for all $\psi \in D_\phi$ is called a well-supported denial for ϕ.

We write $\vdash_{ws} \phi$ ($\vdash_{ws} \neg\phi$) when there is a well-supported proof (denial) for ϕ.

The following theorem states that the model-theoretic and the proof-theoretic views of the least well-supported semantics indeed match.

Theorem 15. Given an OTSS $(R, <)$, let C' be the set of all formulae that have a well-supported proof and I' the set of all formulae that have a well-supported denial. Let (C, P) be the least three-valued stable model of $(R, <)$. Then, $(C', (\Phi_p \setminus I')) = (C, P)$.

4 From Ntyft to Otyft

We assume in the remainder that the TSS's in the ntyft rule format are pure, i.e., only contain variables among the source of the conclusion and targets of the premises. Impure TSS's can be transformed to pure ones (while keeping the TSS in ntyft rule format and preserving the semantics) by making many copies of rules each instantiating the other variables by a closed term [6]. (Hence, there is no expressiveness gap between the ntyft rule format and its pure subset, i.e., all transition relations that can be specified by the ntyft rule format can also be specified by the pure ntyft rule format and vice versa.) Our translation is correct for impure rules, as well but the outcome will not be in the otyft rule format.

Definition 16 (Pure Ntyft to Otyft: Translation Scheme). Given a TSS R in the pure ntyft rule format, its translation to otyft, denoted by $otyft(R)$, is an OTSS $(R', <)$ where $R' \doteq \{r^+, s_{r,j} \mid r \in R, j \in J_r\}$, $< \doteq \{(r^+, s_{r,j}) \mid r \in R, j \in J_r\}$ and for each $r \in R$ of the form $\frac{\{t_i \xrightarrow{l_i} y_i | i \in I_r\} \cup \{t_j \overset{l_j}{\nrightarrow} | j \in J_r\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$, r^+ and $s_{r,j}$ (for each $j \in J_r$) are defined as follows.

$$(r^+)\frac{\{t_i \xrightarrow{l_i} y_i | i \in I_r\}}{f(\overrightarrow{x}) \xrightarrow{l} t} \qquad (s_{r,j})\frac{\{t_j \xrightarrow{l_j} y_j\}}{t_j \xrightarrow{l_j} y_j}$$

In rule $s_{r,j}$, y_j is a fresh variable not appearing in r.

The following theorem states that the diagram depicted in Figure 1.(a) commutes.

$$\begin{tikzcd}
R\text{ (pure ntyft)} \arrow[rr, "\text{Definition 16}"] \arrow[dr, "{[4]}"'] & & otyft(R) \arrow[dl, "\text{Definition 10}"] \\
& \text{3-Valued Stable Model} &
\end{tikzcd}$$

(a)

$$\begin{tikzcd}
(R,<)\text{ (otyft)} \arrow[rr, "\text{Definition 18}"] \arrow[dr, "\text{Definition 10}"'] & & ntyft(R,<) \arrow[dl, "{[4]}"] \\
& \text{3-Valued Stable Model} &
\end{tikzcd}$$

(b)

Fig. 1. Correctness of translations: (a) from ntyft to otyft and (b) from otyft to ntyft

Theorem 17 (Pure Ntyft to Otyft: Correctness). The translation from pure ntyft to otyft preserves its three-valued stable model.

5 From Otyft to NTyft

Definition 18 (Otyft to Ntyft: Translation Scheme). Given an OTSS $(R, <)$ in the otyft rule format, partial function $S_r : R \rightharpoonup \mathcal{I}$, where $\mathcal{I} \doteq \bigcup_{i \in I_r} I_r$, is a selection function for $r \in R$ when for all $s \in higher(r)$ of the form

$\frac{\{t_i \xrightarrow{l_i} y_i | i \in I_s\}}{t \xrightarrow{l} t'}$, it holds that $S_r(s) \in I_s$. (Thus, if $I_s = \emptyset$ for some $s \in \mathit{higher}(r)$, then the set of selection functions for r is empty.)

Given an OTSS $(R, <)$ in the otyft rule format, its translation to ntyft, denoted by $\mathit{ntyft}(R, <)$, is defined as $\{r_S \mid r \in \mathit{tyft}(R),\ S$ is a selection function for $r\}$ where $\mathit{tyft}(R)$ is the subset of R that conforms to the tyft rule format and for each $r \in \mathit{tyft}(R)$ of the form $\frac{\{t_i \xrightarrow{l_i} y_i | i \in I_r\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$, r_S is defined as follows.

$$(r_S)\,\frac{\{t_i \xrightarrow{l_i} y_i | i \in I_r\} \cup \{t_{S(s)} \overset{l_{S(s)}}{\nrightarrow} | s \in \mathit{higher}(r)\}}{f(\overrightarrow{x}) \xrightarrow{l} t}$$

The idea of the above translation is that for each rule r in R, for all rules s placed above r, an arbitrary premise $S(s)$ is negated and included in the premises of r_S. This way, we make sure that r_S is applicable if and only if r is applicable and all rules above it are disabled. We can safely exclude rules that do not conform to the tyft rule format in our translation since their conclusion is among their premises and thus, they do not contribute to the least three-valued well-supported model.

The following theorem states that the diagram depicted in Figure 1.(b) commutes.

Theorem 19 (Otyft to Ntyft: Correctness). The translation from otyft to ntyft preserves its three-valued stable model.

6 Congruence Meta-theorem

As it is shown in [4], for a complete TSS in the ntyft rule format, strong bisimilarity is a congruence. Since our translation (in Section 5) provably preserves the three-valued stable model, we can recast this result to the setting with ordered SOS, as follows.

Theorem 20 (Congruence for Otyft). For a complete OTSS in the otyft rule format, bisimilarity is a congruence.

Note that our only essential addition to the constraints of tyft rule format is the constraint 2.a of Definition 4 (as mentioned before, constraint 2.b can be removed and is only needed to obtain compatibility with the ntyft rule format). The following counter-example shows that constraint 2.a is indeed useful for the purpose of congruence and cannot be dropped.

Example 21.

↓	$\frac{b \xrightarrow{a} y}{b \xrightarrow{a} y}$	$\frac{}{a \xrightarrow{a} b}$	$\frac{}{b \xrightarrow{a} a}$
	$\frac{x \xrightarrow{a} y}{f(x) \xrightarrow{a} a}$		

The above OTSS is complete and it meets all the constraints of Definition 4 except for the constraint 2.a. The C (P) component of the least three-valued stable model of the above OTSS is $\{a \xrightarrow{a} b, b \xrightarrow{a} a, f(a) \xrightarrow{a} a\}$ but $f(b) \xrightarrow{a} a$ is not included in it. Hence, for the above OTSS a and b are bisimilar while $f(a)$ and $f(b)$ are not.

7 Conclusions

In this paper, we presented ways of giving a meaning to ordered SOS specifications. Furthermore, we gave semantics-preserving translations (w.r.t. our chosen notion of semantics) between general ordered SOS and SOS rule formats, namely otyft and ntyft, respectively. Finally, thanks to our semantics-preserving translation, we obtained a congruence meta-result for complete OTSS's in the otyft rule format.

References

1. J. C. M. Baeten, J. A. Bergstra, J.W. Klop. Syntax and defining equations for an interrupt mechanism in process algebra. *Fundamenta Informaticae*, XI(2):127–168, 1986.
2. J. C. M. Baeten, J. A. Bergstra, J.W. Klop, and W. P. Weijland. Term-rewriting systems with rule priorities. *TCS*, 67(2&3):283–301, 1989.
3. B. Bloom, S. Istrail and A.R. Meyer. Bisimulation Can't Be Traced. *JACM*, 42(1):232–268, 1995.
4. R. Bol and J.F. Groote. The meaning of negative premises in transition system specifications. *JACM*, 43(5):863–914, 1996.
5. R.J. van Glabbeek. The meaning of negative premises in transition system specifications II. *JLAP*, 60-61:229–258, 2004.
6. J.F. Groote. Transition system specifications with negative premises. *TCS*, 118(2): 263–299, 1993.
7. M. Hennessy and T. Regan. A process algebra for timed systems. *I&C*, 117(2):221–239, 1995.
8. R. Lanotte and S. Tini. Probabilistic congruence for semistochastic generative processes. In *Proceedings of FOSSACS'05*, volume 3441 of *LNCS*, pages 63–78. Springer, 2005.
9. I. Ulidowski and I.C.C. Phillips. Ordered SOS Rules and Process Languages for Branching and Eager Bisimulations. *I&C*, 178(1):180–213, 2002.

Almost Optimal Strategies in One Clock Priced Timed Games

Patricia Bouyer[1], Kim G. Larsen[2],
Nicolas Markey[1], and Jacob Illum Rasmussen[2]

[1] Lab. Spécification & Vérification, CNRS & ENS Cachan, France
{bouyer, markey}@lsv.ens-cachan.fr
[2] Department of Computer Science, Aalborg University, Denmark
{kgl, illum}@cs.aau.dk

Abstract. We consider timed games extended with cost information, and prove computability of the optimal cost and of ε-optimal memoryless strategies in timed games with one clock. In contrast, this problem has recently been proved undecidable for timed games with three clocks.

1 Introduction

An interesting direction of real-time model checking that has recently received substantial attention is to extend and re-target timed automata technology towards optimal scheduling and planning [1, 15, 9]. In particular, as part of this effort, the notion of priced timed automata [6, 5] has been promoted as a useful extension of the classical model of timed automata [4]. In this extended model each location q is associated with a cost c_q giving the cost of a unit of time spent in q. Thus, each run of a priced timed automaton has an accumulated cost, based on which a variety of optimization problems may be formulated.

Several of the established results concerning priced timed automata are concerned with reachability questions. In [3] cost-bounded reachability was shown decidable. [6] and [5] independently show computability of the cost-optimal reachability for priced (or weighted) timed automata using different adaptations of the so-called region technique. In [13, 15] the notion of priced zone is developed allowing efficient implementation of cost-optimal reachability as witnessed by the competitive tool UPPAAL Cora [16]. Also the problem of computing optimal *infinite* schedules (in terms of minimal limit-ratios) has been shown computable [8]. Finally cost-optimal reachability has been shown decidable in a setting with multiple cost-variables [14].

In this paper we consider the more challenging problem of the computation of *cost-optimal winning strategies* for priced timed *game* automata, *i.e.* a game where the controller tries to win at minimal cost and opponent tries to maximize the cost. Consider the priced timed game with the single clock x depicted in Fig. 1. Here the (circle) locations c_1 and c_2 are controllable whereas (square) locations u_1 and u_2 are uncontrollable with cost-rates being 3, 4, 1 and 1, respectively. All four locations have $x \leq 1$ as invariant. Besides transitions between

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 345–356, 2006.

these four locations, additional transitions are indicated to (triangle) locations for which the optimal costs of winning (for any value of x) are assumed to have already been computed (we call those cost functions *outside cost functions* in the sequel). Obviously, c_1 and c_2 have winning strategies for all values of x by uniformly exiting to their respective outside locations (triangle), c_1^{out} and c_2^{out}. However, this strategy is, clearly, suboptimal for both locations. Alternatively, consider the superior strategy for c_2 depicted in Fig. 2. that guarantees cost no larger than depicted in the corresponding cost function. Then it can be shown that this strategy guarantees the optimal cost.

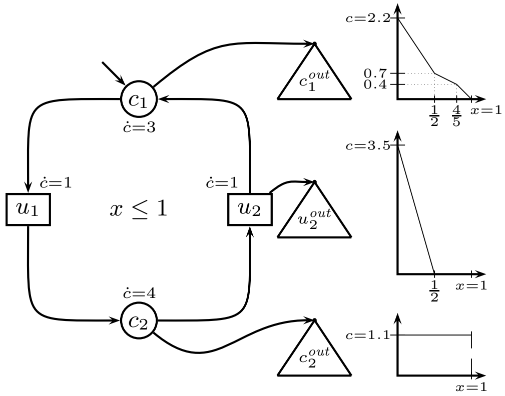
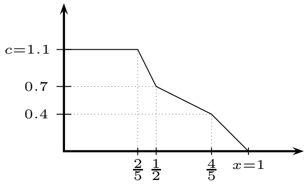


Fig. 1. Sample PTGA with outside cost functions



$$\sigma(c_2,x) = \begin{cases} c_2^{out} & \text{if } 0 \leq x < 2/5 \\ c_2 & \text{if } 2/5 \leq x < 1/2 \\ u_2 & \text{if } 1/2 \leq x \leq 1 \end{cases}$$

Fig. 2. An optimal strategy in c_2, and the associated cost function

In [12] the problem of computing cost-optimal winning strategies has been studied and shown computable for *acyclic* priced timed games. Furthermore, in [11] it is proven that computing optimal winning strategies for one-clock PTGA with stopwatch cost (*i.e.* cost are either zero or one) is decidable. [2] and [10] provide partial solutions to the general case of non-acyclic games: under the assumption of certain non-Zenoness behaviour of the underlying priced timed automata it is shown that it suffices only to consider strategies guaranteed to win within some given number k of steps, or alternatively to unfold the given game k times and reduce the problem to solving an acyclic game. To see how restricted these results are, it may be observed that the priced timed game in Fig. 1 does not belong to any of the above classes. In fact, in [11] it has recently been shown that the problem of determining cost-optimal winning strategies for priced timed games is not computable. Most recently, it has been shown that this negative result holds even for priced timed (game) automata with no more than three clocks [7].

In this paper we completely solve the computation of cost-optimal winning strategies for arbitrary priced timed (game) automata *with one clock*: we offer an algorithm for computing optimal costs, explain why optimal strategies need not always exist, whereas memoryless ε-optimal strategies exist and can be computed.

2 Definitions

We write x for the (unique) clock variable, and $\mathcal{X} = \{x\}$. A clock constraint for clock x is an expression of the form $x \in I$ where I is an interval over the reals with integer (or infinite) bounds which can have strict or non-strict bounds. As a shortcut, we may use expressions like $x \geq 5$ instead of $x \in [5, +\infty[$. The set of all clock constraints is denoted $\mathcal{B}(\mathcal{X})$. That a valuation $v \colon \mathcal{X} \to \mathbb{R}_+$ satisfies a clock constraint g is defined in a natural way (v satisfies $x \in I$ whenever $v(x) \in I$), and we then write $v \models g$. We denote by v_0 the valuation that assigns zero to clock x, by $v + t$ ($t \in \mathbb{R}_+$) the valuation that assigns $v(x) + t$ to $x \in \mathcal{X}$.

A *cost function* is a piecewise affine function $f \colon \mathbb{R}_+ \to \mathbb{R}_+ \cup \{+\infty\}$ with negative slopes. We also require that if $\{+\infty\} \in f((n, n+1))$ for some integer n, then $f((n, n+1)) = \{+\infty\}$, and that f is continuous over all intervals $(n, n+1)$. We write CF for the set of all cost functions.

We define an extended notion of priced timed games, with outside cost functions and urgent locations. Those extra features will be needed throughout the proof. A 1-*clock priced timed game with outside cost functions* (PTG_f for short) is a tuple $G = (Q_c, Q_u, Q_f, Q_{\mathrm{urg}}, Q_{\mathrm{init}}, f_{\mathrm{goal}}, T, \eta, P)$ where

- Q_c is a finite set of *controllable locations*, Q_u is a finite set of *uncontrollable locations*. Those sets are disjoint, and we define $Q = Q_c \cup Q_u$;
- Q_f is the set of final locations (it is disjoint from Q).
- $Q_{\mathrm{urg}} \subseteq Q_u$ indicates urgent uncontrollable locations;
- $Q_{\mathrm{init}} \subseteq Q$ is the set of *initial* locations;
- $f_{\mathrm{goal}} \colon Q_f \to \mathrm{CF}$ assigns to each final location a cost function;
- $T \subseteq Q \times \mathcal{B}(\mathcal{X}) \times 2^{\mathcal{X}} \times (Q \cup Q_f)$ is the set of *transitions*;
- $\eta \colon Q \to \mathcal{B}(\mathcal{X})$ defines the *invariants* of each location;
- $P \colon Q \cup T \to \mathbb{N}$ is the *cost (or price) function.*

Standard (1-clock) priced timed games [2,10] are PTG_f with $Q_{\mathrm{urg}} = \varnothing$ and, for any $q \in Q_f$, $f_{\mathrm{goal}}(q)(\mathbb{R}_+) = \{0\}$ or $\{+\infty\}$.

In the following, G will always refer to a PTG_f, and we will not always rewrite the corresponding tuple. Similarly, G' will denote a PTG_f whose components are "primed".

We assume (w.l.o.g., see [6]) that the clock is bounded, *i.e.*, there exists an integer M such that for every location $q \in Q$, $\eta(q) \Rightarrow x \leq M$.

Let G be a PTG_f. The semantics of G is given as a labeled timed transition system $\mathcal{T} = (S, S_{\mathrm{init}}, \to)$ where $S \subseteq (Q \cup Q_f) \times \mathbb{R}_+$ is the set of states[1], $S_{\mathrm{init}} = Q_{\mathrm{init}} \times \{v_0\}$ is the set of initial states, and the transitions relation $\to \subseteq S \times \mathbb{R}_+ \times S$ is defined as:

1. *(discrete transition)* $(q, v) \xrightarrow{c} (q', v')$ if $q \notin Q_f$ and there exists $(q, g, R, q') \in T$ such that $v(x) \models g$, $v' = [R \leftarrow 0]v$, $v'(x) \models \eta(q')$, and $c = P(q, g, R, q')$;
2. *(delay transition)* $(q, v) \xrightarrow{c} (q, v + t)$ if $q \notin Q_{\mathrm{urg}} \cup Q_f$, and $\forall 0 \leq t' \leq t$, $v + t' \models \eta(q)$, and $c = t \cdot P(q)$.

[1] Formally, $S \subseteq (Q \cup Q_f) \times (\mathbb{R}_+)^{\mathcal{X}}$, but we identify v with $v(x)$ here.

A *run* of G is a (finite) path in the underlying transition system. Given $T, U \subseteq S$, we write[2] $\mathsf{Run}_G(T, U)$ for the set of runs of G issued from $t \in T$ and ending in $u \in U$. Given a run ϱ and a position $v \in \varrho$ along that run, the prefix of ϱ ending in v is denoted by $\varrho_{|v}$. A run is *maximal* if either it is infinite, or no discrete transition is possible (even after a delay transition). A maximal run is *accepting* if it is finite and ends in a final location. Let $\varrho = s_0 \xrightarrow{c_0} s_1 \xrightarrow{c_1} \cdots \xrightarrow{c_{n-1}} s_n$ be a run. Its cost, denoted $\mathsf{cost}(\varrho)$, is either $\sum_{i=0}^{n-1} c_i$ if ϱ is not accepting, or $\sum_{i=0}^{n-1} c_i + f_{\text{goal}}(q_n)(v_n(x))$, where $(q_n, v_n) = s_n$ if ϱ is accepting. An accepting run is *winning* if it has finite cost.

Example. Reconsider the example depicted in Fig. 1. Here, a sample winning run is $\varrho = (c_1, 0) \xrightarrow{0} (u_1, 0) \xrightarrow{0.4} (u_1, 0.4) \xrightarrow{0} (c_2, 0.4) \xrightarrow{0.4} (c_2, 0.5) \xrightarrow{0} (c_2^{out}, 0.5)$ which has cost $\mathsf{cost}(\varrho) = 0.4 \times 1 + 0.1 \times 4 + f_{\text{goal}}(c_2^{out})(0.5) = 1.9$. □ A *strategy* is then a function $\sigma \colon \mathsf{Run}_G(Q \times \mathbb{R}_+, Q_c \times \mathbb{R}_+) \to \{\lambda\} \cup Q \cup Q_f$. Informally, a strategy tells in all controllable locations, what has to be done, and the special symbol λ indicates to delay. A strategy σ is *memoryless* if $\sigma(\varrho) = \sigma(\varrho')$ as soon as ϱ and ϱ' end in the same state.

Let σ be a strategy in G, and ϱ_0 a run in G ending in (q_0, x_0). A run $\varrho = (q_0, x_0) \xrightarrow{c_0} (q_1, x_1) \xrightarrow{c_1} \cdots \xrightarrow{c_{n-1}} (q_n, x_n)$ is a (σ, ϱ_0)-run if for all delay- (or discrete-) transitions $(q_i, x_i) \xrightarrow{c_i} (q_{i+1}, x_{i+1})$ where $q_i \in Q_c$, we have

– $\forall x \in [x_i, x_{i+1}[,\ \sigma(\varrho_0 \cdot \varrho_{|x}) = \lambda$,
– $\sigma(\varrho_0 \cdot \varrho_{|x_i}) = q_{i+1}$.

where $\varrho_0 \cdot \varrho$ denotes the (usual) concatenation. In that case, we say that ϱ is *compatible* with σ after ϱ_0 (or that it is an *outcome* of σ after ϱ_0). We write $\mathsf{Run}_{G,\sigma}(\varrho_0, U)$ for the set of such runs ending in U.

A strategy σ is said *accepting* after (run) ϱ_0 whenever all maximal runs in $\mathsf{Run}_{G,\sigma}(\varrho_0)$ are accepting. If a strategy is not accepting from ϱ_0, we set its cost in G after ϱ_0, $\mathsf{Cost}_G(\sigma, \varrho_0)$, to $+\infty$. Otherwise its cost in G after ϱ_0 is given as: $\mathsf{Cost}_G(\sigma, \varrho_0) = \sup\{\mathsf{cost}(\varrho) \mid \varrho \in \mathsf{Run}_{G,\sigma}(\varrho_0, Q_f \times \mathbb{R}_+)\}$. Obviously, for any two runs ϱ_0 and ϱ_1 ending in (q, x), the sets $\{\mathsf{Cost}_G(\sigma, \varrho_0) \mid \sigma \text{ strategy in } G\}$ and $\{\mathsf{Cost}_G(\sigma, \varrho_1) \mid \sigma \text{ strategy in } G\}$ are equal. An accepting strategy σ after ϱ_0 is *winning* if $\mathsf{Cost}_G(\sigma, \varrho_0)$ is finite. We define for every state s of G, the *optimal cost* of winning from s as $\inf\{\mathsf{Cost}_G(\sigma, \varrho_0) \mid \sigma \text{ strategy in } G\}$ for some run ϱ_0 ending in s. We denote it $\mathsf{OptCost}_G(s)$. If $\mathsf{OptCost}_G(s) < +\infty$, the state s is said *winning* in G. In that case, for every $\varepsilon > 0$, for every run ϱ_0 ending in s, there exists a winning strategy σ s.t. $\mathsf{OptCost}_G(s) \leq \mathsf{Cost}_G(\sigma, \varrho_0) < \mathsf{OptCost}_G(s) + \varepsilon$, and we say that σ is *ε-optimal* from ϱ_0. A strategy σ is *optimal* from ϱ_0 if $\mathsf{Cost}_G(\sigma, \varrho_0) = \mathsf{OptCost}_G(s)$ where ϱ_0 ends in state s.

A strategy σ in G is *(ε, N)-acceptable* (with $\varepsilon > 0$, and $N \in \mathbb{N}$) whenever: (1) it is memoryless, (2) it is ε-optimal, (3) there exist N (consecutive) intervals $(I_i)_{1 \leq i \leq N}$ partitioning $[0, 1]$ such that for every location q, for every $1 \leq i \leq N$, for every integer $\alpha < M$, the function $x \mapsto \mathsf{Cost}_G(\sigma, (q, x))$ is affine on every interval $\alpha + I_i$, and the function $x \mapsto \sigma(q, x)$ is constant on $\alpha + I_i$.

[2] In the sequel, we might omit the subscripts G when they are clear from the context.

3 Main Result

The main result of this paper is that optimal cost is computable and that almost-optimal memoryless strategies always exist and can be effectively computed. This is summarized by the following theorem:

Theorem 1. *Let G be a PTG$_f$. Then for every location q in G, the function $x \mapsto \mathit{OptCost}_G((q,x))$ is computable and piecewise-affine. Moreover, for every $\varepsilon > 0$, there exists (and we can effectively compute) a strategy σ in G such that σ is memoryless and ε-optimal in every state.*

We will even prove a stronger result, which is that there exists $N \in \mathbb{N}$ such that for every $\varepsilon > 0$, we can effectively compute an (ε, N)-acceptable strategy σ. The rest of this paper is devoted to a proof of this result.

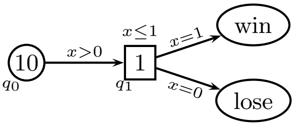


Fig. 3. A game with no optimal strategy

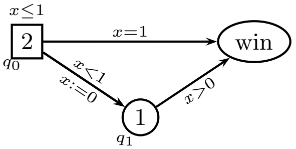


Fig. 4. A game where optimal strategies require memory

There are PTG$_f$ for which no optimal strategies exist, as exemplified by Fig. 3: from q_0, the optimal cost is 1, but a winning strategy consists in delaying in q_0 for some duration $\delta > 0$, yielding a cost of $1 + 9\delta$. This is why we compute, in the general case, ε-optimal strategies. In the same way, as witnessed by Fig 4, it might be the case that optimal strategies exist but require some amount of memory: in the example of Fig 4, state $(q_0, x = 0)$ is winning with optimal cost 2, but no memoryless strategy can achieve that cost for sure.

4 Simplifying Transformations

In this section, we first explain how to restrict to simpler games while preserving the same optimal costs, and we then show how we can inductively compute optimal cost on those simpler games. We also explain how to compute almost-optimal strategies for those simpler games, and how to "lift" those strategies to the original game.

Our transformations consist in two steps: (i) we restrict to PTG$_f$ where the clock is bounded by 1 (denoted $[0,1]$-PTG$_f$) (Section 4); (ii) we restrict to a $[0,1]$-PTG$_f$ without resetting transition (Section 4). For each transformation, we prove that:

- the optimal cost in each state of the original game is identical to the optimal cost in some corresponding state in the transformed game,

– we can derive an ε-optimal strategy in the original game from some ε'-optimal strategy in the transformed game.

Section 5 is then devoted to computing the optimal cost and an almost-optimal strategy in the simpler game. For the sake of simplicity, we assume here that there are no discrete costs on transitions. A slight adaptation of the transformation for removing resets can be given for handling discrete costs as well.

Restricting to a PTG$_f$ bounded by 1. The idea of this construction is to reset the clock each time it reaches 1, and to record in the discrete structure what should be the real integer part of the value of the clock (the clock will only store the fractional part of its real value).

Let G be a PTG$_f$. We build another PTG$_f$ G' such that for every $q' \in Q'$, $\eta'(q')$ implies $0 \leq x \leq 1$, and G' is correct for computing optimal cost, in a sense which will be made clear later.

As we have assumed that PTG$_f$ are bounded, we set M the constant bounding G, and we define:

$$\begin{cases} Q'_x &= \{q_{[\alpha,\alpha+1]} \mid q \in Q_x \text{ and } 0 \leq \alpha < M\} \text{ for every } x \in \{c, u, f, \mathrm{urg}\} \\ Q'_{\mathrm{init}} &= \{q_{[0,1]} \mid q \in Q_{\mathrm{init}}\} \end{cases}$$

The set of transitions T' is composed of the following transitions (if g is a guard, $g - \alpha$ denotes the same guard translated by $-\alpha$):

$$\begin{cases} q_{[\alpha,\alpha+1]} \xrightarrow{(g-\alpha)\cap(0\leq x<1)} q'_{[\alpha,\alpha+1]} & \text{if } (q \xrightarrow{g} q') \in T \text{ and } \alpha+1 < M \\ q_{[M-1,M]} \xrightarrow{(g-\alpha)\cap(0\leq x\leq 1)} q'_{[M-1,M]} & \text{if } (q \xrightarrow{g} q') \in T \\ q_{[\alpha,\alpha+1]} \xrightarrow[x:=0]{(g-\alpha)\cap(0\leq x<1)} q'_{[0,1]} & \text{if } (q \xrightarrow[x:=0]{g} q') \in T \text{ and } \alpha+1 < M \\ q_{[M-1,M]} \xrightarrow[x:=0]{(g-\alpha)\cap(0\leq x\leq 1)} q'_{[0,1]} & \text{if } (q \xrightarrow[x:=0]{g} q') \in T \\ q_{[\alpha-1,\alpha]} \xrightarrow[x:=0]{x=1} q_{[\alpha,\alpha+1]} & \text{if } 0 < \alpha < M \end{cases}$$

The invariant η' is defined by $\eta'(q_{[\alpha,\alpha+1]}) = (0 \leq x \leq 1) \wedge (\eta(q) - \alpha)$ if $q \in Q$.

The cost function P' is defined by $P'(q_{[\alpha,\alpha+1]}) = P(q)$. The function f'_{goal} is defined by $f'_{\mathrm{goal}}(q_{[\alpha,\alpha+1]})(x) = f_{\mathrm{goal}}(q)(x+\alpha)$ for every $0 \leq x \leq 1$.

Note that all guards and invariants of G' are included in $[0,1]$, we say that G' is a $[0,1]$-*PTG$_f$* .

We define f the function which maps every state (q, x) of G onto the state $(q_{[\alpha,\alpha+1]}, x - \alpha)$ of G' such that $0 \leq x - \alpha \leq 1$ and $x < M$ integer implies $x = \alpha$. We now state the following correctness result.

Proposition 2. *For every state (q, x) in G, $\mathsf{OptCost}_G(q, x) = \mathsf{OptCost}_{G'}(f(q, x))$. Moreover, for every $\varepsilon > 0$ and $N \in \mathbb{N}$, given an (ε, N)-acceptable strategy in G', we can compute an (ε, N)-acceptable strategy in G, and vice-versa.*

Removing resetting transitions from SCCs. We have restricted to games with a single clock. A strong property of this model is that each time a resetting transition is taken, then the *very same state* is visited (because the valuation is each time v_0). The construction for removing resetting transitions takes advantage of this property.

Let G be a PTG$_f$ with n resetting transitions. From the previous reduction, we may assume that all the invariants and guards in G imply that $0 \leq x \leq 1$. We build a PTG$_f$ G', made of $n+1$ copies of G, such that no strongly connected component (SCC for short) of G' contains a resetting transition.

We thus define $Q'_c = Q_c \times \{0, ..., n\}$, $Q'_u = Q_u \times \{0, ..., n\}$, and $Q'_f = (Q_f \times \{0, ..., n\}) \cup \{r\}$. A location $(q, i) \in Q'_u$ is urgent iff $q \in Q_{\text{urg}}$. We let $Q'_{\text{init}} = Q_{\text{init}} \times \{0\}$. The outside cost functions are given by $f'_{\text{goal}}((q, i)) = f_{\text{goal}}(q)$, and $f_{\text{goal}}(r) = +\infty$. The invariant is given by $\eta'((q, i)) = \eta(q)$ for $q \in Q$. Transitions are defined as follows:

$$\begin{cases} ((q,i) \xrightarrow{g} (q',i)) \in T' & \text{if } (q \xrightarrow{g} q') \in T \text{ and } i \leq n \\ ((q,i) \xrightarrow[x:=0]{g} (q',i+1)) \in T' & \text{if } (q \xrightarrow[x:=0]{g} q') \in T \text{ and } i < n \\ ((q,n) \xrightarrow{g} r) \in T' & \text{if } (q \xrightarrow[x:=0]{g} q') \in T \text{ and } i = n \end{cases}$$

Last, we set $P'((q, i)) = P'(q)$ for every $q \in Q$, and the price of each transition of T' defined above is the price of the corresponding transition in T.

Proposition 3. *For every state (q, x) in the game G, $\mathsf{OptCost}_G((q, x))$ equals $\mathsf{OptCost}_{G'}(((q, 0), x))$. Moreover, for every $\varepsilon' > 0$ and $N' \in \mathbb{N}$, given an (ε', N')-acceptable strategy in G', we can compute a $(2\varepsilon', N')$-acceptable strategy in G.*

We have thus reduced our problem to computing optimal cost and almost-optimal winning strategies in G'. In G', this can be done by first computing it in the nth copy of G, and then in the $(n-1)$th copy of G, etc.

5 Computing Almost-Optimal Strategies

We have restricted our problem to $[0, 1]$-PTG$_f$ without resets. We can also easily restrict to such PTG$_f$ containing only one SCC: if we can compute the optimal costs and an (ε, N)-acceptable strategy on an SCC, we will be able to handle the general case by working first on the deepest SCC, and then replace it by the corresponding outside function (and an (ε, N)-acceptable strategy).

Thus, we now assume that we only work on a $[0, 1]$-PTG$_f$ without resets and based on an SCC. We prove the following result, which will imply Theorem 1.

Theorem 4. *Let G be a $[0, 1]$-PTG$_f$ without reset such that $(Q_c \cup Q_u, T)$ is an SCC (or contains only one location). Then:*

H1. $\mathsf{OptCost}_G(q, x)$ is computable for every $q \in Q$ and every $x \in [0, 1]$;

H2. for every location $q \in Q$, $x \in [0, 1] \mapsto \mathsf{OptCost}_G(q, x)$ is a cost function whose finitely many segments either have slope $-c$ where $c \in P(Q)$, or are fragments of the outside cost functions of G;

H3. there exists an integer N such that, for any $\varepsilon > 0$, we can compute an (ε, N)-acceptable strategy in G for every $q \in Q$ and every $x \in [0,1]$.

The rest of this section is devoted to the proof of this theorem, which is by induction on the number of non-urgent locations in G. First we prove the base case of the induction, that is when the game is only composed of urgent locations, or of a single controllable location.

- Proving properties H1 and H2 in the case where G contains only one location is handled straightforwardly, by combining the outside cost functions of G with the cost rate of the location. Property H3 requires more care. Let q be a (controllable) location with a bunch of outside cost functions $\{f_{\text{goal}}(q') \mid q' \in Q_f\}$. Define the function $s\colon x \to \min\{f_{\text{goal}}(q', x) \mid q' \in Q_f\}$. Then $\mathsf{OptCost}_G(q,x) = \inf_{x \leq x' \leq 1} P(q) \cdot (x' - x) + s(x')$. Let $\varepsilon > 0$. We then define the strategy σ as follows:

$$\sigma(q,x) = \begin{cases} q' \text{ if } \mathsf{OptCost}_G(q,x) = f_{\text{goal}}(q')(x) \\ \lambda \text{ if } \mathsf{OptCost}_G(q,x) < s(x) \text{ and either } s(1) < +\infty \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{or } x \leq 1 - \varepsilon/(2P(q)) \\ q' \text{ if } \mathsf{OptCost}_G(q,x) < s(x),\ s(1) = +\infty,\ 1 - \varepsilon/(2P(q)) < x < 1, \\ \quad \text{and } \lim_{x \to 1^-} f_{\text{goal}}(q')(x) = \lim_{x \to 1^-} s(x) \end{cases}$$

  It is not difficult to check that σ is (ε, N)-acceptable for some N which is independent of ε.
- The case where G contains an SCC with only urgent (thus uncontrollable) locations is also straightforward, since the opponent can force the game to never reach a final location, and the optimal cost is then $+\infty$. If the game is composed of a single urgent location, then this is also easy.

We now assume that G is an SCC composed of at least two locations, n of which are non-urgent. We select one of the non-urgent locations having least cost, and denote it with $q_{\min}$, and, depending on the nature (controllable or not) of $q_{\min}$, we explain how we prove that Theorem 4 holds for G if it holds for SCCs having at most $(n-1)$ non-urgent locations.

Case: $q_{\min}$ is controllable. For handling this case, we will prove that the rough intuition that there is no need to delay twice in $q_{\min}$, but we better delay longer in $q_{\min}$ is indeed correct.

From the game G, we construct a game G', made of two copies of G, such that each SCC of the new game contains one location less (see Fig. 5). We define $Q'_c = (Q_c \setminus \{q_{\min}\}) \times \{0,1\} \cup \{q_{\min}\}$, $Q'_u = Q_u \times \{0,1\}$, $Q'_f = Q_f \times \{0,1\} \cup \{r\}$, $Q'_{\text{urg}} = Q_{\text{urg}} \times \{0,1\}$, $Q'_{\text{init}} = Q_{\text{init}} \times \{0\}$, $f'_{\text{goal}}((q,i)) = f_{\text{goal}}(q)$ if $q \in Q_f$, and $f'_{\text{goal}}(r) = +\infty$, $\eta'((q,i)) = \eta(q)$, $\eta'(q_{\min}) = \eta(q_{\min})$, $P'((q,i)) = P(q)$ for every $(q,i) \in Q'_c \cup Q'_u$. The set of transitions is

$$\begin{aligned} T' = \{&(q,i) \xrightarrow{g,R} (q',i) \mid q \xrightarrow{g,R} q', \text{and } q, q' \neq q_{\min}\} \\ \cup\ \{&(q,0) \xrightarrow{g,R} q_{\min},\ (q,1) \xrightarrow{g,R} r \mid (q \xrightarrow{g,R} q_{\min}) \in T\} \\ \cup\ \{&q_{\min} \xrightarrow{g,R} (q',1) \mid (q_{\min} \xrightarrow{g,R} q') \in T\}. \end{aligned}$$

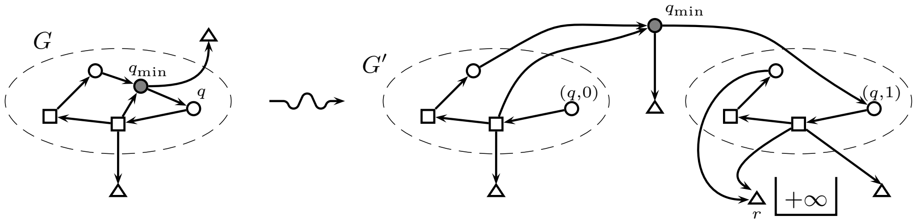


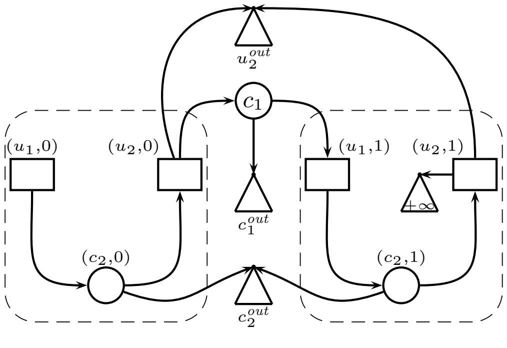
Fig. 5. Case $q_{\min}$ (in grey) controllable

We prove the following lemma, which establishes properties H1 and H2.

Lemma 5. *For every $(q,x) \in (Q \setminus \{q_{\min}\}) \times [0,1]$, we have $\mathsf{OptCost}_G(q,x) = \mathsf{OptCost}_{G'}((q,0),x)$. For every $x \in [0,1]$, $\mathsf{OptCost}_G(q_{\min},x) = \mathsf{OptCost}_{G'}(q_{\min},x)$.*

It remains to prove property H3. We fix the integer N' for G'. We fix some $\varepsilon > 0$, and take $\varepsilon' = \frac{\varepsilon}{3}$. We take σ' an (ε', N')-acceptable strategy in G'. We then define σ as follows:

$$\sigma(q,x) = \begin{cases} \sigma'((q,1),x) & \text{if } \mathsf{Cost}_{G'}(\sigma',((q,1),x)) \le \mathsf{OptCost}_{G'}(q_{\min},x) \\ \sigma'((q,0),x) & \text{otherwise} \end{cases} \tag{1}$$



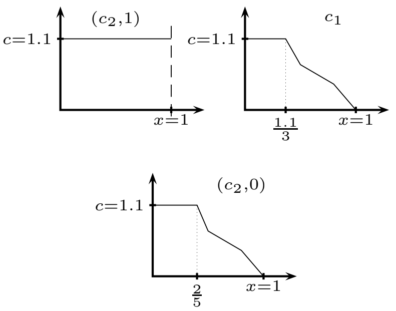


Fig. 6. Running example after unwinding

Fig. 7. Optimal costs

Example. Returning to the running example of Fig. 1 with u_1 and u_2 urgent, performing the above transformation with respect to c_1 gives the PTG$_f$ depicted in Fig. 6. The optimal cost functions are depicted in Fig. 7 and the resulting winning strategy for c_2 is, according to (1), the strategy of $(c_2,1)$ when $x \le \frac{1.1}{3}$ and $(c_2,0)$ otherwise. □

Obviously, the strategy σ is memoryless. We need to establish that the function $x \mapsto \mathsf{Cost}_G(\sigma,(q,x))$ consists of at most N pieces, and that σ is ε-optimal.

Proposition 6. *Strategy σ is winning and there exists a fixed (independant of ε) integer N such that σ is (ε,N)-acceptable.*

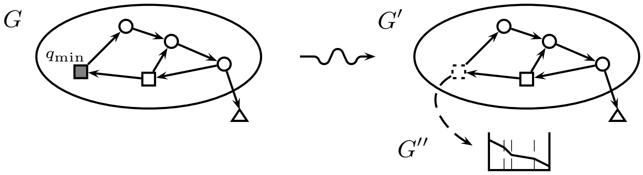


Fig. 8. When it is uncontrollable, $q_{\min}$ is made urgent (in dash line here)

Case: $q_{\min}$ is uncontrollable. The intuition is that the opponent will prefer delays in other locations than $q_{\min}$ whenever possible. We attempt to enforce this by a transformation of the game where location $q_{\min}$ is urgent, as depicted in Fig. 8. Formally, given a $[0, 1]$-PTG$_f$ without resets G, we define G' with $Q'_{\text{urg}} = Q_{\text{urg}} \cup \{q_{\min}\}$ and $Q'_u = Q_u \backslash \{q_{\min}\}$.

Obviously enough, since we restrict the possible moves for the opponent in G', we have for every state (q, x), $\mathsf{OptCost}_{G'}(q, x) \leq \mathsf{OptCost}_G(q, x)$.

However, the converse inequality is not correct over $[0, 1]$, and we will need a more complex construction to handle this case. We now explain how to iteratively compute the optimal costs in G. Fig. 9 gives an overview of the computation described below.

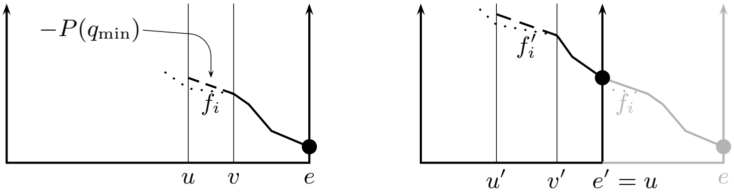


Fig. 9. Successive computations when $q_{\min}$ is uncontrollable

Clearly, we can compute $\mathsf{OptCost}_G(q_{\min}, 1)$ (indeed, $\mathsf{OptCost}_G(q_{\min}, 1) = \mathsf{OptCost}_{G'}(q_{\min}, 1)$, since when $x = 1$, time cannot elapse any more and the same moves are available in G' and in G). This initializes our iterative computation.

Now, assume we can compute $\mathsf{OptCost}_G(q_{\min}, e)$ for some $e \in [0, 1]$. We can apply the induction hypotheses H1—H3 to G'. In particular, $f : x \in [0, e] \mapsto \mathsf{OptCost}_{G'}(q_{\min}, x)$ is a cost function satisfying the requirements of item H2. Writing f_1, ..., f_n for the successive affine functions constituting f, we pick the smallest index i such that for every $j > i$, function f_j has slope less than or equal to $-P(q_{\min})$. If $i > 0$, we note $[u, v]$ the domain of f_i (see Fig. 9).

Lemma 7. *If $i = 0$, for all $(q, x) \in Q \times [0, e]$, $\mathsf{OptCost}_G(q, x) = \mathsf{OptCost}_{G'}(q, x)$. If $i > 0$, for all $(q, x) \in Q \times [v, e]$, $\mathsf{OptCost}_G(q, x) = \mathsf{OptCost}_{G'}(q, x)$.*

We now explain how to compute $\mathsf{OptCost}_G(q_{\min}, x)$ for $x \in [u, v]$; we prove the following lemma:

Lemma 8. *If $i > 0$, then for all $(q, x) \in Q \times [u, v]$, we have $\mathsf{OptCost}_G(q_{\min}, x) = (v - x)P(q_{\min}) + f(v)$.*

The optimal cost in states (q, x) with $x \in [u, v]$ can then be computed by considering the PTG$_f$ G'', restricted to $x \in [u, v]$, and obtained from G' by making $q_{\min}$ a goal location with cost function equal to $x \mapsto \mathsf{OptCost}_G(q_{\min}, x)$, which is then viewed as an outside cost function, see Fig. 9.

We can then repeat the procedure above on the interval $[0, u]$ (*i.e.* by setting $e = u$): compute $f' \colon x \mapsto \mathsf{OptCost}_{G'}(q_{\min}, x)$ with $x \in [0, u]$, select an interval $[u', v']$ where f'_i has slope larger than or equal to $-P(q_{\min})$, and so on, replace that part with an affine function with slope $-P(q_{\min})$, and continue with the interval $[0, u']$. We now explain why this process terminates: since they have slopes strictly greater than $-P(q_{\min})$, f_i and f'_i are fragments of outside cost functions, according to hypothesis H2. If they have different slopes, then they are obviously parts of two different fragments of outside cost functions. If they have the same slopes, then they are fragments of two different parts of outside cost functions, since they are joined by affine functions with slopes less than (or equal to $-P(q_{\min})$). Since there are only finitely many affine functions constituting the outside cost functions, our procedure terminates.

At each step of the procedure above, we can also compute (ε, N)-acceptable strategies, and merge them.

6 Conclusion

In this paper we have proven that optimal cost for arbitrary priced timed games with one clock is a computable problem, and that ε-optimal memoryless strategies may effectively be obtained. The complexity of our procedure is quite high, running in 3-EXPTIME, while the best known lower bound for this problem is PTIME. Our future works of course include tightening these bounds.

As a consequence of our result it may be shown that the iterative semi-algorithm proposed in [10] always terminates for priced timed games with one clock. Cost functions cost^i_G are inductively defined, which for any location $q \in Q$ and any clock value v, give the optimal cost of winning from the state (q, v) within at most i steps (we count the number of steps in a run ρ by the number of delay-and-action fractions). Now Theorem 4 ensures that we can find a fixed N such that for *any* $\varepsilon > 0$ we can compute an (ε, N)-acceptable strategy. In particular this guarantees that we can find ε-optimal strategies which are guaranteed to win within $N \cdot |Q|$ steps for any $\varepsilon > 0$. Consequently, $\langle \mathsf{cost}^i_G \rangle_{i=1}^{\infty}$ (the semi-algorithm of [10]) converges after at most $N \cdot |Q|$ iterations to the optimal cost of winning. A prototype implementation of this iterative algorithm is available at http://www.lsv.ens-cachan.fr/~markey/1ptga/.

As future work we would like to determine what happens with priced timed games using two clocks, but this seems really difficult as our approach heavily relies on the fact that there is only one clock.

Acknowledgement. Work partly supported by ACI Cortos, a program of the French ministry of research.", and this footnote would be pointed from both names "Patricia Bouyer" and "Nicolas Markey.

References

1. Y. Abdeddaïm, E. Asarin, and O. Maler. Scheduling with timed automata. 2006.
2. R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability in weighted timed games. In *Proc. 31st Intl Coll. Automata, Languages and Programming (ICALP'04)*, LNCS 3142, p. 122–133. Springer, 2004.
3. R. Alur, C. Courcoubetis, and Th. A. Henzinger. Computing accumulated delays in real-time systems. In *Proc. 5th Intl Conf. Computer Aided Verification (CAV'93)*, LNCS 697, p. 181–193. Springer, 1993.
4. R. Alur and D. Dill. A theory of timed automata. *Theor. Comp. Science*, 126(2):183–235, 1994.
5. R. Alur, S. La Torre, and G. J. Pappas. Optimal paths in weighted timed automata. In *Proc. 4th Intl Workshop Hybrid Systems: Computation and Control (HSCC'01)*, LNCS 2034, p. 49–62. Springer, 2001.
6. G. Behrmann, A. Fehnker, Th. Hune, K. G. Larsen, P. Pettersson, J. Romijn, and F. Vaandrager. Minimum-cost reachability for priced timed automata. In *Proc. 4th Intl Workshop Hybrid Systems: Computation and Control (HSCC'01)*, LNCS 2034, p. 147–161. Springer, 2001.
7. P. Bouyer, Th. Brihaye, and N. Markey. Improved undecidability results on weighted timed automata. *Inf. Proc. Letters*, (5):188–194, June 2006.
8. P. Bouyer, E. Brinksma, and K. G. Larsen. Staying alive as cheaply as possible. In *Proc. 7th Intl Workshop Hybrid Systems: Computation and Control (HSCC'04)*, LNCS 2993, p. 203–218. Springer, 2004.
9. P. Bouyer, E. Brinksma, and K. G. Larsen. Optimal infinite scheduling for multi-priced timed automata. *Form. Meth. in Syst. Design*, 2006. To appear.
10. P. Bouyer, F. Cassez, E. Fleury, and K. G. Larsen. Optimal strategies in priced timed game automata. In *Proc. 24th Conf. Foundations of Software Technology & Theoretical Computer Science (FST&TCS'04)*, LNCS 3328, p. 148–160. Springer, 2004.
11. Th. Brihaye, V. Bruyère, and J.-F. Raskin. On optimal timed strategies. In *Proc. 3rd Intl Conf. Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, LNCS 3821, p. 49–64. Springer, 2005.
12. S. La Torre, S. Mukhopadhyay, and A. Murano. Optimal-reachability and control for acyclic weighted timed automata. In *Proc. 2nd IFIP Intl Conf. Theoretical Computer Science (IFIPTCS'02)*, IFIP Conf. Proc. 223, p. 485–497. Kluwer, 2002.
13. K. G. Larsen, G. Behrmann, E. Brinksma, A. Fehnker, Th. Hune, P. Pettersson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *Proc. 13th Intl Conf. Computer Aided Verification (CAV'01)*, LNCS 2102, p. 493–505. Springer, 2001.
14. K. G. Larsen and J. I. Rassmussen. Optimal conditional reachability for multi-priced timed automata. In *Proc. 8th Intl Conf. Foundations of Software Science and Computation Structures (FoSSaCS'05)*, LNCS 3441, p. 234–249. Springer, 2005.
15. J. I. Rasmussen, K. G. Larsen, and K. Subramani. Resource-optimal scheduling using priced timed automata. In *Proc. 10th Intl Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'04)*, LNCS 2988, p. 220–235. Springer, 2004.
16. UPPAAL CORA. `http://www.cs.aau.dk/~behrmann/cora/`, Jan. 2006.

Expressivity Properties of Boolean BI Through Relational Models

Didier Galmiche[⋆] and Dominique Larchey-Wendling[⋆⋆]

LORIA – CNRS[⋆] – UHP Nancy 1[⋆⋆]
Campus Scientifique, BP 239
54 506 Vandœuvre-lès-Nancy, France

Abstract. In this paper, we study Boolean BI Logic (BBI) from a semantic perspective. This logic arises as a logical basis of some recent separation logics used for reasoning about mutable data structures and we aim at proposing new results from alternative semantic foundations for BBI that seem to be necessary in the context of modeling and proving program properties. Starting from a Kripke relational semantics for BBI which can also be viewed as a non-deterministic monoidal semantics, we first show that BBI includes some S4-like modalities and deduce new results: faithful embeddings of S4 modal logic, and then of intuitionistic logic (IL) into BBI, despite of the classical nature of its additive connectives. Moreover, we provide a logical characterization of the observational power of BBI through an adequate definition of bisimulation.

1 Introduction

Separation logics are logics for reasoning about mutable data structures [9,11,14] in which pre- and post-conditions are written in a logic enriched with specific forms of conjunction or implication. They are mainly based on the logic of Bunched Implications (BI) which combines standard (additive) intuitionistic implication $\rightarrow$ and conjunction $\wedge$ (additive connectives) and linear (intuitionistic) implication $-\!\!*$ and conjunction $*$ (multiplicative connectives) [12]. Actually, they mainly deal with Boolean BI (BBI) that is the version of BI in which the additive connectives are classical. Compared to BI, BBI needs further investigations from both semantic and proof-theoretic points of view. Recently we have proposed results about propositional BI: new semantics (based on relations or partially defined monoids) [8], labelled calculi and related proof-search methods from which decidability and finite model property have been proved [6].

Our aim is to obtain similar results for BBI, in order to provide new proof-theoretical foundations for this logic but also for some computational models of BBI like separation and spatial logics [2,14]. Even if the difference with BI is mainly the classical nature of additives, we cannot directly derive such results from those of BI, for instance a (complete) based-on monoid semantics like in BI [8] or in classical BI pointer logic [9]. Therefore it seems important to study new semantic foundations of BBI that initially has an algebraic model called Boolean BI-algebra [12]. In this context, we start from a Kripke relational semantics for BBI, that is proved sound and complete, and also provide an equivalent semantics based on non-deterministic monoids. The first ternary-relation

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 357–368, 2006.

models for BBI, defined by Yang [16], are based on the notion of maximally consistent sets. Their definitions and proofs strongly use classical negation and are tailored towards BBI. Our semantics, that is equivalent, is in the continuation of our works about relational models of (intuitionistic) BI [8] and does not deal with negation. It might be suitable to consider open problems like, for instance, the existence of a (deterministic) based-on monoid semantics for BBI? Even if we can define, from this semantics, labelled calculi and thus prove their completeness for BBI, it would remain to study properties of propositional BBI, like decidability and finite model property. Our relational models for BBI, that extend those for BI, seem to provide good foundations for such a study. As a consequence of the soundness property, we propose as central contributions embeddings of modal logic S4, and then of intuitionistic logic IL into BBI. The later could be surprising despite of the classical nature of its additive connectives. These embeddings have consequences on the use of BBI from proof-search and complexity perspectives. To complete these results we also provide a logical characterization of its observational power through an adequate definition of bisimulation.

2 Boolean BI

Boolean BI, denoted by BBI, is a mixed logic like BI [12] that has some computational models like separation and spatial logics [2,14]. It is built on a set Var of propositional variables combined using *additive* connectives of classical propositional logic ($\wedge$, $\vee$, $\rightarrow$, $\neg$, $\bot$ and $\top$) and *linear* connectives of multiplicative linear logic ($*$, $-\!\!*$ and I).

$$\frac{A \vdash \neg\neg B}{A \vdash B}\ [\text{RAA}]$$

Provability in BBI is defined in [12] by adding the rule of *reductio ad absurdum* denoted [RAA] to the natural deduction calculus of BI.[1] In this paper, like in [16], we rather adopt a Hilbert deduction system to define provability in BBI. We only recall here the axioms and rules that characterize BBI. First, we choose any (finite) set of axioms for the classical part of BBI among the axiom sets for classical propositional logic.[2] We add to it the following axioms for the linear part: $A \rightarrow (\mathsf{I} * A)$; $(\mathsf{I} * A) \rightarrow A$;$A * B) \rightarrow (B * A)$; $(A * (B * C)) \rightarrow ((A * B) * C)$. All these axioms should be considered as *schemes*, i.e., we consider the closure of the set of axioms under (uniform) substitutions. Moreover, we have the following Hilbert *deduction rules* for BBI:

$$\frac{\vdash A \qquad \vdash A \rightarrow B}{\vdash B}\ [\text{MP}] \qquad \frac{\vdash A \rightarrow C \qquad \vdash B \rightarrow D}{\vdash (A * B) \rightarrow (C * D)}\ [*]$$

$$\frac{\vdash A \rightarrow (B -\!\!* C)}{\vdash (A * B) \rightarrow C}\ [-\!\!*_1] \qquad \frac{\vdash (A * B) \rightarrow C}{\vdash A \rightarrow (B -\!\!* C)}\ [-\!\!*_2]$$

The [MP]-rule is the usual *modus ponens* and the three other rules $[*]$, $[-\!\!*_1]$ and $[-\!\!*_2]$ hold for BI and BBI. Compared to BI, the additive axioms of BBI are those of classical logic instead of intuitionistic logic. So the set of "classical" axioms of BBI contains a form of *reductio ad absurdum*, like for example $\neg\neg A \rightarrow A$. An algebraic model for this

[1] With $\neg A$ defined as $\neg A \equiv A \rightarrow \bot$.

[2] Such axioms could be for example $A \rightarrow (A \vee B), A \rightarrow (B \rightarrow (A \wedge B)), \ldots$.

system is called a *Boolean* BI*-algebra*. We denote by $A \simeq B$ the logical equivalence of A and B (both $\vdash A \rightarrow B$ and $\vdash B \rightarrow A$ are deducible from the axioms).

Proposition 1. *The following logical equivalences hold in* BI *and* BBI*:*

$$\begin{array}{rcl@{\qquad}rcl} \bot * A & \simeq & \bot & (A \vee B) * C & \simeq & (A * C) \vee (B * C) \\ \bot \mathrel{-\!\!*} A & \simeq & \top & (A \vee B) \mathrel{-\!\!*} C & \simeq & (A \mathrel{-\!\!*} C) \wedge (B \mathrel{-\!\!*} C) \\ A \mathrel{-\!\!*} \top & \simeq & \top & A \mathrel{-\!\!*} (B \wedge C) & \simeq & (A \mathrel{-\!\!*} B) \wedge (A \mathrel{-\!\!*} C) \end{array}$$

Computational models of BBI, like BI's pointer logic (PL), are used for reasoning about mutable data structures [9] and we aim at studying them in a proof-theoretic perspective from their semantics [7]. Starting from our results on BI [6,8] we need first to study relational models for BBI.

3 A Kripke Relational Semantics for BBI

Before to study semantics of BBI, we emphasize the relationships between the notions of *non-deterministic* monoid and so-called *relational frame*.

3.1 Non-deterministic Monoids and Relational Semantics

Let us consider a set $\mathcal{M}$. We denote by $\mathcal{P}(\mathcal{M})$ the powerset of $\mathcal{M}$, i.e. its set of subsets. A binary function $\circ : \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{P}(\mathcal{M})$ is naturally extended to a binary operator on $\mathcal{P}(\mathcal{M})$ by $X \circ Y = \bigcup\{x \circ y \mid x \in X, y \in Y\}$ for any subsets X, Y of $\mathcal{M}$. Using this extension, we can identify an element m of $\mathcal{M}$ with the singleton set $\{m\}$ and derive the equations $m \circ X = \{m\} \circ X$ and $a \circ b = \{a\} \circ \{b\}$.

Definition 1. *A* non-deterministic monoid *is a triple* $(\mathcal{M}, \circ, \mathsf{e})$ *where* $\mathsf{e} \in \mathcal{M}$ *and* $\circ : \mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{P}(\mathcal{M})$ *for which the following conditions hold:*
1. $\forall a \in \mathcal{M}, \mathsf{e} \circ a = \{a\}$ *(identity)*
2. $\forall a, b \in \mathcal{M}, a \circ b = b \circ a$ *(commutativity)*
3. $\forall a, b, c \in \mathcal{M}, a \circ (b \circ c) = (a \circ b) \circ c$ *(associativity)*[3]

The term *non-deterministic* is introduced in order to emphasize the fact that the composition $a \circ b$ may yield not only one but several results including the possible incompatibility of a and b in which case $a \circ b = \emptyset$. If $(\mathcal{M}, \times, 1)$ is a commutative monoid then, defining $a \circ b = \{a \times b\}$ and $\mathsf{e} = 1$ induces a non-deterministic monoid structure on $\mathcal{M}$. Partial monoids can also be represented using the empty set $\emptyset$ as the result of undefined compositions. We claim that the notion of non-deterministic monoid is an extension of the notion of partial commutative monoid. Then, these models generalize the (associative and commutative) tree based models [5] or the process based models [3] of separation logics. We establish an algebraic link between non-deterministic monoids and Boolean-BI algebras.

Proposition 2. *Let the triple* $(\mathcal{M}, \circ, \mathsf{e})$ *be a non-deterministic monoid,* $(\mathcal{P}(\mathcal{M}), \subseteq, \circ)$ *is a quantale and also a complete boolean algebra.*

[3] The axiom of associativity should be understood using the extension of $\circ$ to $\mathcal{P}(\mathcal{M})$.

Using the isomorphism between $\mathcal{M} \times \mathcal{M} \longrightarrow \mathcal{P}(\mathcal{M})$ and $\mathcal{M} \times \mathcal{M} \times \mathcal{M} \longrightarrow \mathbf{2} = \{\mathsf{false} < \mathsf{true}\}$, we define a ternary relation $\triangleright \subseteq \mathcal{M} \times \mathcal{M} \times \mathcal{M}$ by: $a,b \triangleright c$ iff $c \in a \circ b$.

Then we can also consider non-deterministic monoids as relational frames.

Definition 2. *A* relational frame *is a triple* $(\mathcal{M}, \triangleright, \mathsf{e})$ *where* e *is an element of* $\mathcal{M}$ *and* $\triangleright$ *a ternary relation on* $\mathcal{M}$ *satisfying, for all* $a,b,c,d \in \mathcal{M}$*:*
1. $e,a \triangleright b$ *iff* $a = b$ *(identity)*
2. $a,b \triangleright c$ *iff* $b,a \triangleright c$ *(commutativity)*
3. if $\exists k\,(a,k \triangleright d$ *and* $b,c \triangleright k)$ *then* $\exists p\,(a,b \triangleright p$ *and* $p,c \triangleright d)$ *(associativity)*

The relation $m,a \triangleright b$ can be read in both directions: "the composition of m and a yields b" or "b is decomposable into m and a." We claim that relational frames can model process calculi and resource calculi or a combination of both like in [13]. The two notions of non-deterministic monoid and relational frame are in fact completely isomorphic. In the following, we will rather use the relational frame notion.

Moreover, from Proposition 2, it is clear that non-deterministic monoids (or equivalently relational frames) induce Boolean BI algebras on the powerset of their career.

3.2 A Relational Semantics for BBI

Let $(\mathcal{M}, \triangleright, \mathsf{e})$ be a relational frame and $v : \mathsf{Var} \longrightarrow \mathcal{P}(\mathcal{M})$ be a *valuation*, i.e. an interpretation of propositional variables. We define, by induction on formulae, a *forcing relation* $\Vdash$ between elements of $\mathcal{M}$ and formulae of BBI:

$$\begin{array}{rll@{\qquad}rll}
m \Vdash \mathsf{I} & \text{iff} & m = \mathsf{e} & m \Vdash X & \text{iff} & m \in v(X) \\
m \Vdash \bot & \text{iff} & \text{never} & m \Vdash A \vee B & \text{iff} & m \Vdash A \text{ or } m \Vdash B \\
m \Vdash \top & \text{iff} & \text{always} & m \Vdash A \wedge B & \text{iff} & m \Vdash A \text{ and } m \Vdash B \\
m \Vdash \neg A & \text{iff} & m \nVdash A & m \Vdash A \rightarrow B & \text{iff} & m \nVdash A \text{ or } m \Vdash B
\end{array}$$

$$\begin{array}{rll}
m \Vdash A * B & \text{iff} & \exists a,b \text{ s.t. } a,b \triangleright m \text{ and } a \Vdash A \text{ and } b \Vdash B \\
m \Vdash A \mathrel{-\!\!*} B & \text{iff} & \forall a,b\ (m,a \triangleright b \text{ and } a \Vdash A) \text{ implies } b \Vdash B
\end{array}$$

Theorem 1 (Soundness). *Let* $(\mathcal{M}, \triangleright, \mathsf{e})$ *be a relational frame and* v *be a valuation. If a formula* A *of* BBI *is provable then for any element* m *of* $\mathcal{M}$*,* $m \Vdash A$ *holds.*

Proof. Let us fix a relational frame $(\mathcal{M}, \triangleright, \mathsf{e})$ and a valuation $v : \mathsf{Var} \longrightarrow \mathcal{P}(\mathcal{M})$. Since the interpretation of the additive connectives of BBI is the standard Kripke interpretation of classical propositional connectives, all theorems of classical logic are forced by all elements of $\mathcal{M}$. Moreover the rule [MP] preserves forcing since it is the standard *modus ponens* rule of classical logic. We only have to check that the linear axioms are forced and also that the three deduction rules $[*]$, $[\mathrel{-\!\!*}_1]$ and $[\mathrel{-\!\!*}_2]$ preserve forcing.

Let us check axiom 4 (section 2). Let m be such that $m \Vdash A * (B * C)$. We prove that $m \Vdash (A * B) * C$. By definition of the forcing relation, there exist a,k s.t. $a,k \triangleright m$ and $a \Vdash A$ and $k \Vdash B * C$. So there exist $b,c \triangleright k$ s.t. $b \Vdash B$ and $c \Vdash C$. Thus $a,k \triangleright m$ and $b,c \triangleright k$ holds. By associativity of the $\triangleright$ relation, there exists p s.t. $a,b \triangleright p$ and $p,c \triangleright m$. Since $a,b \triangleright p$, we deduce $p \Vdash A * B$ and since $p,c \triangleright m$, we deduce $m \Vdash (A * B) * C$.

Now let us check the deduction rule $[\mathrel{-\!\!*}_1]$. Suppose that for any element m of $\mathcal{M}$, $m \Vdash A \rightarrow (B \mathrel{-\!\!*} C)$ holds. Let k be an element of $\mathcal{M}$ such that $k \Vdash A * B$ holds. Let us prove that $k \Vdash C$ holds. Since $k \Vdash A * B$, there exist a,b s.t. $a,b \triangleright k$ and $a \Vdash A$ and $b \Vdash B$. Since $a \Vdash A$ holds and $a \Vdash A \rightarrow (B \mathrel{-\!\!*} C)$ holds (by instantiation of the hypothesis), then $a \Vdash B \mathrel{-\!\!*} C$ holds. But since $a,b \triangleright k$ holds, by definition of the forcing relation, we deduce $k \Vdash C$.

We now study the completeness of this relational semantics by extending techniques we used for completeness of the relational semantics of BI [8]. We define a term model based on the *Lindenbaum construction* and the *prime filters* of this boolean algebra. We denote by $\mathcal{L}$ the set of classes of logically equivalent formulae and these classes by the letters $a,b,c...$ The class of a formula A is denoted $[A] = \{B \mid A \simeq B\}$. The $\simeq$ equivalence relation is a congruence and the logical connectives induce algebraic operators on the Lindenbaum algebra. An order relation is defined between classes by $[A] \leqslant [B]$ iff $\vdash A \rightarrow B$ is provable and $(\mathcal{L}, \leqslant)$ has the structure of a boolean algebra with least element $0 = [\bot]$ and greatest element $1 = [\top]$, each classical connective inducing a corresponding boolean operator. We introduce $\mathsf{i} = [\mathsf{I}]$ as the class of the monoidal unit.

Filters and prime filters. The *upward closure* of a subset X of $\mathcal{L}$ is defined by $\uparrow X = \{k \in \mathcal{L} \mid \exists x \in X, x \leqslant k\}$. A *filter* F of $\mathcal{L}$ is a non-empty ($1 \in F$) upward closed ($\uparrow F = F$) and meet-stable ($\forall x,y \in F, x \wedge y \in F$) subset of $\mathcal{L}$. If x is an element of $\mathcal{L}$ then $\uparrow x$ defined by $\uparrow x = \{k \in \mathcal{L} \mid x \leqslant k\}$ is the least filter containing x. $\uparrow 0 = \mathcal{L}$ is the greatest filter.

A *prime filter* F_p of $\mathcal{L}$ is a filter which is *proper* ($0 \notin F_p$) and satisfies $\forall a,b \in \mathcal{L}, a \vee b \in F_p$ implies ($a \in F_p$ or $b \in F_p$). Let us recall the following result: since $\mathcal{L}$ is a boolean algebra, the prime filters are exactly the maximal proper filters of $\mathcal{L}$ [4].

Proposition 3. *Let F_p, G_p be prime filters of $\mathcal{L}$. If $F_p \subseteq G_p$ then $F_p = G_p$.*

We denote by $\mathbb{F}$ (resp. $\mathbb{F}_p$) the set of filters (resp. prime filters) of $\mathcal{L}$. They are ordered by inclusion $\subseteq$ and, by Proposition 3, the order on $\mathbb{F}_p$ is flat. We define a (commutative) monoidal operation on $\mathbb{F}$ by $A \bullet B = \uparrow\{a * b \mid a \in A \text{ and } b \in B\}$. Then $(\mathbb{F}, \subseteq, \bullet, \uparrow\mathsf{i})$ is an *ordered commutative monoid*. The greatest filter $\uparrow 0$ is an absorbing element of $\bullet$.[4]

Definition 3. *A* prime predicate $\varphi : \mathbb{F} \longrightarrow \mathbf{2} = \{\mathsf{false} < \mathsf{true}\}$ *satisfies*
1. $\bigwedge_k \varphi(F_k) \leqslant \varphi(\bigcup_k F_k)$ *for any chain* $(F_k)_{k \in I}$.
2. $\varphi(F \cap G) \leqslant \varphi(F) \vee \varphi(G)$ *for any filters* F, G.
3. $\varphi(\uparrow 0) = \mathsf{false}$.

Let us give two examples of prime predicates. Let $x < 1$ be an element of $\mathcal{L}$. Then the map $F \mapsto x \notin F$ is a prime predicate. Let $A \in \mathbb{F}$ and $H_p \in \mathbb{F}_p$ then $F \mapsto A \bullet F \subseteq H_p$ is also a prime predicate.[5]

Lemma 1 (prime extension). *If φ if a prime predicate and F a filter such that* $\varphi(F) = \mathsf{true}$, *then there exists a prime filter F_p extending F ($F \subseteq F_p$) and such that* $\varphi(F_p) = \mathsf{true}$.

This lemma is proved using Zorn's lemma and expresses that a filter satisfying a prime predicate can be extended to a prime filter satisfying the same predicate.

[4] So for any filters F, G, if $0 \in F$ then $0 \in F \bullet G$.

[5] This property involves the distributivity of $*$ over $\vee$ (see Proposition 1).

Corollary 1. *We have the two following results:*
1. Let $x \in \mathcal{L}$ and $F \in \mathbb{F}$ s.t. $x \notin F$. There exists $F_p \in \mathbb{F}_p$ s.t. $F \subseteq F_p$ and $x \notin F_p$. In particular, if $x < 1$, there exists $F_p \in \mathbb{F}_p$ s.t. $x \notin F_p$.
2. Let $A, B \in \mathbb{F}$ and $H_p \in \mathbb{F}_p$ s.t. $A \bullet B \subseteq H_p$. There exist $A_p, B_p \in \mathbb{F}_p$ s.t. $A \subseteq A_p$, $B \subseteq B_p$ and $A_p \bullet B_p \subseteq H_p$.

Term models with one unit. Indeed, the set $\mathbb{F}_p$ of prime filters cannot be used directly as a model of BBI because several extensions of the unit I exist. We have to select a particular one to obtain a model with a unique unit, problem also studied in [16].

Definition 4. *Let I_p, F_p be prime filters. I_p is a* unit *if* $\mathsf{i} \in I_p$. *I_p is a* unit of *F_p if I_p is a unit and $I_p \bullet F_p \subseteq F_p$.*

Since $\mathsf{i} \in I_p$, for any filter F we have $F \subseteq I_p \bullet F$. Consequently if I_p is a unit of F_p then the identity $I_p \bullet F_p = F_p$ holds.

Proposition 4. *Let I_p, I'_p be two units and F_p be a prime filter, we have*
1. I_p is a unit of I_p;
2. $0 \notin I_p \bullet I'_p$ if and only if $I_p = I'_p$;
3. $0 \notin I_p \bullet F_p$ if and only if I_p is a unit of F_p.

Proposition 5. *Every prime filter has a unique unit.*

Proposition 6. *Let A_p, B_p and C_p be prime filters. If $A_p \bullet B_p \subseteq C_p$ holds then A_p, B_p and C_p share the same unit.*

We now can build a term model with a unique unit. Let us fix a unit I_p. Among the primer filters, we only consider those having I_p for unit. Let $\mathcal{M} = \{F_p \in \mathbb{F}_p \mid I_p \text{ is a unit of } F_p\}$. We define the ternary relation $\triangleright$ on $\mathcal{M}$ by $A_p, B_p \triangleright C_p$ iff $A_p \bullet B_p \subseteq C_p$. We also define a valuation $v : \mathsf{Var} \longrightarrow \mathcal{P}(\mathcal{M})$ by $F_p \in v(X)$ iff $[X] \in F_p$. We interpret propositional variables with the valuation v and obtain a forcing relation $\Vdash$.

Lemma 2. *The triple $(\mathcal{M}, \triangleright, I_p)$ is a relational frame, in which we use the previously defined forcing relation $\Vdash$. Then, for any formula A of* BBI *and any prime filter F_p of $\mathcal{M}$, $F_p \Vdash A$ iff $[A] \in F_p$.*

Theorem 2 (Completeness). *If A is not provable in* BBI, *then there exists a relational frame which is a counter-model of A.*

Proof. Let A be not provable in BBI. Let a be its class $[A]$ in $\mathcal{L}$. Then $a < 1$. So by Corollary 1, there exists a prime filter F_p such that $a \notin F_p$. Let I_p be the unit of F_p and $\mathcal{M}$ be relation frame associated to I_p according to Lemma 2. Then $F_p \in \mathcal{M}$ since I_p is the unit of F_p. Moreover $[A] = a \notin F_p$ and thus $F_p \nVdash A$.

Compared to the relational semantics of Yang [16], that is based on maximally consistent sets, our semantics extends the one we defined for (intuitionistic) BI [8] and can be seen as more abstract. It generalizes previous models for separation logics for trees and processes. From such a semantics we could define a tableau method for BBI and proved its completeness but in order to study decidability and finite model properties for BBI we need to deeper analyze the resolution of relational constraints.

4 Embeddings of S4 and IL into BBI

In this section, we exploit the relational semantics of BBI, mainly its soundness, in addition to the definition of a S4-like modality of BBI in order to faithfully embed the modal logic S4, and then the intuitionistic logic IL, into BBI.

4.1 A S4-Like Modality in BBI

We introduce the denotation $\Box A$ as an abbreviation of $\top \mathrel{-\!\!*} A$. Given a relational frame $(\mathcal{M}, \triangleright, \mathsf{e})$, we define the relation $\preccurlyeq$ between elements of $\mathcal{M}$ by $a \preccurlyeq b$ if there exists $m \in \mathcal{M}$ such that $m, a \triangleright b$. It is easy to verify that $\preccurlyeq$ is a *preorder* on the set $\mathcal{M}$, i.e., a reflexive and transitive relation. Moreover the Kripke interpretation of the $\Box$ operator is expressed by: $m \Vdash \Box A$ iff $\forall k, m \preccurlyeq k$ implies $k \Vdash A$.

Then $\Box A$ is Kripke interpreted the same way as in S4. Let us check now if the axioms of S4 are theorems of BBI.

Proposition 7. *The three axioms $\Box(A \to B) \to (\Box A \to \Box B)$, $\Box A \to A$ and $\Box A \to \Box\Box A$ of* S4 *are provable in* BBI. *If A is provable in* BBI*, then $\Box A$ is provable in* BBI.

Proof. We give a proof of $\Box A \to A$. Let $K_1 \equiv (\top \mathrel{-\!\!*} A) * \mathsf{I}$, $K_2 \equiv (\top \mathrel{-\!\!*} A) * \top$. $(\top \mathrel{-\!\!*} A) \to (\top \mathrel{-\!\!*} A)$ is a (classical) axiom of BBI. So by rule $[\mathrel{-\!\!*}_1]$, $K_2 \to A \equiv ((\top \mathrel{-\!\!*} A) * \top) \to A$ is provable. Moreover $\mathsf{I} \to \top$ is a (classical) tautology of BBI and then, by rule $[*]$, $K_1 \to K_2 \equiv ((\top \mathrel{-\!\!*} A) * \mathsf{I}) \to ((\top \mathrel{-\!\!*} A) * \top)$ is provable. As $(\top \mathrel{-\!\!*} A) \to K_1 \equiv (\top \mathrel{-\!\!*} A) \to ((\top \mathrel{-\!\!*} A) * \mathsf{I})$ is an axiom of BBIwe get, by combining $(\top \mathrel{-\!\!*} A) \to K_1$ with $K_1 \to K_2$ and $K_2 \to A$, a proof of $\Box A \to A$.

We now prove the deduction rule. Let A be a provable formula of BBI. Then $\top \to A$ is provable. Moreover $(\top * \top) \to \top$ is (classical) tautology of BBI. Combining those two, $(\top * \top) \to A$ is provable and then by rule $[\mathrel{-\!\!*}_2]$, $\top \to (\top \mathrel{-\!\!*} A)$ is provable. Moreover $\top$ is a (classical) axiom and thus, by rule [MP], $\top \mathrel{-\!\!*} A$ is provable, i.e., $\Box A$ is provable.

As a corollary to this result, we define a mapping from formulae of S4 to formulae of BBI built on the same set Var of propositional variables. Let $A \mapsto A^{\Box}$ be recursively defined by the following equations:

$$\begin{array}{rclrcll} (\neg A)^{\Box} & = & \neg A^{\Box} & K^{\Box} & = & K & \text{for } K \in \mathsf{Var} \cup \{\bot, \top\} \\ (\Box A)^{\Box} & = & \top \mathrel{-\!\!*} A^{\Box} & (A \otimes B)^{\Box} & = & A^{\Box} \otimes B^{\Box} & \text{for } \otimes \in \{\wedge, \vee, \to\} \end{array}$$

Corollary 2. *If A is a provable formula of* S4 *then $A^{\Box}$ is provable in* BBI.

Proof. By Proposition 7, all the (specific) axioms and deductions rules of S4 are also derivable in BBI. The other rule of S4 (which is [MP]) and the other axioms of S4 are those of classical propositional logic, which is a part of BBI.

4.2 From Trees to Relational Frames

A *partial order* $\leqslant$ is a reflexive, antisymmetric and transitive relation. Two elements a and b are *upper bounded* when they have a common upper bound m such that $a \leqslant m$ and $b \leqslant m$. Two elements a and b are *comparable* if either $a \leqslant b$ or $b \leqslant a$.

Definition 5. A tree $(\mathcal{T},\leqslant,r)$ *is a partial order where r is the least element of* $\mathcal{T}$*. Moreover any two upper bounded elements of* $\mathcal{T}$ *are comparable.*

Theorem 3. *If A is a formula of* S4 *which is not provable, then there exists a tree* $(\mathcal{T},\leqslant,r)$ *such that* $r \not\Vdash A$*.*

Proof. We recall the main argument of the proof given in [1] (pages 59–63). Since A is not provable, it has a counter-model $(\mathcal{Q},\leqslant)$ in the class of preorders. For some element $r \in \mathcal{Q}$, the property $r \not\Vdash A$ holds. Consider the set $\mathcal{S}$ of *finite increasing sequences* of the form $(r = a_0, a_1, \ldots, a_n)$ for $n \geqslant 0$. It is ordered by the *prefix order* between sequences and thus $\mathcal{S}$ is a tree with root (r). The mapping $(a_0, a_1, \ldots, a_n) \mapsto a_n$ from $\mathcal{S}$ to $\mathcal{Q}$ is a *surjective bounded morphism* so it preserves the forcing relation. Thus $(r) \not\Vdash A$ in $\mathcal{S}$.

Let $(\mathcal{T},\leqslant,r)$ be a tree. Then the max operator is a partial commutative monoidal operator with unit r. We build a ternary relation on $\mathcal{T}$ by:

$$a,b \triangleright m \text{ iff } a \text{ and } b \text{ are comparable and } m = \max\{a,b\}$$

Proposition 8. $(\mathcal{T},\triangleright,r)$ *is a relational frame and the preorder* $\preccurlyeq$ *induced by* $\triangleright$ *matches* $\leqslant$*, i.e.,* $\preccurlyeq\, = \,\leqslant$*.*

Proof. Since r is the neutral element of $\mathcal{T}$, the identity axiom is obvious. Commutativity is also obviously verified. Let us check associativity. If $a,k \triangleright d$ and $b,c \triangleright k$ hold, then b and c are comparable and $k = \max\{b,c\}$. Then d is an upper bound of a, b and c. Since, $k \in \{b,c\}$ and $d \in \{a,k\}$, then $d \in \{a,b,c\}$. Thus $d = \max\{a,b,c\}$. Since $\mathcal{T}$ is a tree and a and b are upper bounded by d, then a and b are comparable[6] and let $p = \max\{a,b\}$. Then $a,b \triangleright p$ and $p,c \triangleright d$. We conclude that $\triangleright$ is associative. If $a \preccurlyeq b$ there exists m s.t. $m,a \triangleright b$. Then $b = \max\{m,a\}$ and we obtain $a \leqslant b$. Conversely if $a \leqslant b$ then $r,a \triangleright b$ and thus $a \preccurlyeq b$ holds. Consequently the identity $\preccurlyeq\, = \,\leqslant$ holds.

Theorem 4. *If A is not provable in* S4*, then* $A^{\Box}$ *is not provable in* BBI*.*

Proof. Since A is not provable in S4, by Theorem 3, there exist a (potentially infinite) tree $(\mathcal{T},\leqslant,r)$ and a valuation $v : \mathsf{Var} \longrightarrow \mathcal{P}(\mathcal{T})$ s.t. $r \not\Vdash_{\mathsf{S4}} A$. We consider the associated relational frame $(\mathcal{T},\triangleright,r)$ and use the same valuation v. By Proposition 8, the identity $\preccurlyeq\, = \,\leqslant$ holds. By a structural induction on F, formula of S4, we prove that for any $m \in \mathcal{T}$, $m \Vdash_{\mathsf{S4}} F$ iff $m \Vdash_{\mathsf{BBI}} F^{\Box}$. Then, in particular, $r \not\Vdash_{\mathsf{BBI}} A^{\Box}$. Then $(\mathcal{T},\triangleright,r)$ associated to v is a counter-model of $A^{\Box}$. By soundness, we deduce that $A^{\Box}$ is not provable in BBI.

A direct consequence of the faithful embedding $A \mapsto A^{\Box}$ is the following: it is well known that propositional intuitionistic logic IL can be faithfully embedded into S4 by prefixing with a $\Box$ all variables $X \mapsto \Box X$ and implications $(A \to B) \mapsto \Box(A \to B)$ while preserving the rest of the structure of the formula. Thus combining both embeddings we have the following result:

Theorem 5. *There exist faithful embeddings of* S4 *and* IL *into* BBI*.*

[6] Here the fact that $\mathcal{T}$ is a tree is required. The max operator would not necessarily be associative if $\mathcal{T}$ was only a partial order or preorder.

This result is surprising because we could naively think that BBI with its classical propositional connectives has a "classical" nature. Moreover, such embeddings have an impact on proof-search in BBI. In particular, if BBI is decidable as we still hope to prove it in further works, its complexity is at least polynomial-space complete (the complexity of IL [15] and S4). Even if it is complete w.r.t. to partial orders or trees, S4, does not have the finite model property for these models. However, S4 has the finite model property for preorders [1]. This point emphasizes the importance of the right tuning of axioms when seeking the finite model property and could be a hint to a finer axiomatization of relational semantics. In further work we will study if there exists a faithful embedding of multiplicative intuitionistic linear logic MILL into BBI.

5 BBI and Bisimulation in Relational Frames

In this section, we deal with the formulae of BBI in order to distinguish elements of relational frames. We provide a characterization of the *observational power* of BBI: it is the ω-limit denoted $\sim_\omega$ of the transfinite decreasing sequence leading to the greatest bisimulation (see [10]) denoted $\sim$. Then, we discuss further conditions under which the identity $\sim = \sim_\omega$ would hold. We consider the Lindenbaum algebra $\mathcal{L}$ of BBI. Unlike what we have done before, we do not distinguish between a formula A and its class of logical equivalence $[A]$. So we write $A = B$ when we have $A \simeq B$.

5.1 BBI in Finite Slices

Let δ be the function defining the weight of binary logical connectives: $\delta(\vee) = \delta(\wedge) = \delta(\rightarrow) = 0$ and $\delta(*) = \delta(-\!\!*) = 1$. The *rank* of a formula A, denoted $\mathsf{rank}(A)$, is defined by induction on the structure of A as follows:

$$\mathsf{rank}(\neg A) = \mathsf{rank}(A) \text{ and } \mathsf{rank}(K) = 0 \text{ for } K \in \mathsf{Var} \cup \{\bot, \top, \mathsf{I}\}$$
$$\mathsf{rank}(A \otimes B) = \max\{\mathsf{rank}(A), \mathsf{rank}(B)\} + \delta(\otimes) \text{ for } \otimes \in \{\vee, \wedge, \rightarrow, *, -\!\!*\}$$

Then an additive connective preserves the rank while a linear connective increases the rank by one. This notion of rank is not the same as in [5] but it serves the same purpose: to cut BBI into finite slices. The rank of a class of logically equivalent formulae is the least rank of its representatives (i.e. its elements). We denote by $\mathcal{L}_r$ the subset of $\mathcal{L}$ composed of (classes of) formulae of rank at most r. Obviously, since boolean (additive) connectives preserve the rank, $\mathcal{L}_r = \{A \in \mathcal{L} \mid \mathsf{rank}(A) \leqslant r\}$ is a sub-boolean algebra of $\mathcal{L}$. Then $\mathcal{L}_0$ contains all the propositional variables of Var and the multiplicative unit I.

Let $\mathcal{K}$ be a subset of $\mathcal{L}$. The *sub-boolean algebra generated by* $\mathcal{K}$, denoted $\mathcal{B}(\mathcal{K})$, is the least subset of $\mathcal{L}$ containing $\mathcal{K} \cup \{\bot, \top\}$ and closed under the boolean operators $\vee$, $\wedge$, $\rightarrow$ and $\neg$. It is clear that $\mathcal{B}(\cdot)$ is a *closure operator* on $\mathcal{L}$. Moreover formulae of rank 0 cannot contain the $*$ or $-\!\!*$ connectives, so any formula of $\mathcal{L}_0$ is a boolean combination of atomic formulae and $\mathcal{L}_0 = \mathcal{B}(\mathsf{Var} \cup \{\mathsf{I}\})$.

Proposition 9. *If $\mathcal{K}$ is a finite subset of $\mathcal{L}$ then $\mathcal{B}(\mathcal{K})$ is finite.*

Proof. Suppose $\mathcal{K} = \{K_1, \ldots, K_n\}$ and let $\mathcal{X} = \{X_1, \ldots, X_n\}$ be a set of (distinct) variables. We denote by $\mathcal{B}_\mathcal{X}$ the (finite) boolean algebra freely generated by $\mathcal{X}$. There is a unique boolean algebra homomorphism $\varphi : \mathcal{B}_X \longrightarrow \mathcal{L}$ such that $\forall i\ \varphi(X_i) = K_i$. Its image is the least sub-boolean algebra of $\mathcal{L}$ containing $\mathcal{K}$: $\varphi(\mathcal{B}_X) = \mathcal{B}(\mathcal{K})$. Since $\mathcal{B}_X$ is finite, then $\mathcal{B}(\mathcal{K})$ is finite.

Let $\mathcal{K}$ be a finite subset of $\mathcal{L}$. We define a mapping $\overline{(\cdot)} : \mathcal{P}(\mathcal{K}) \longrightarrow \mathcal{L}$ from subsets of $\mathcal{K}$ to $\mathcal{L}$ by: $\overline{\Gamma} = \bigwedge\{A \mid A \in \Gamma\} \wedge \bigwedge\{\neg A \mid A \in \mathcal{K} - \Gamma\}$
It is clear that for any $\Gamma \in \mathcal{P}(\mathcal{K})$, $\overline{\Gamma}$ is an element of $\mathcal{B}(\mathcal{K})$. In fact, the direct image of the mapping $\overline{(\cdot)} : \mathcal{P}(\mathcal{K}) \longrightarrow \mathcal{B}(\mathcal{K})$ is either Min or $\mathsf{Min} \cup \{\bot\}$, where Min is the set of *minimal elements* of $\mathcal{B}(\mathcal{K}) - \{\bot\}$.

Proposition 10. *For $A \in \mathcal{K}$, the identity $A = \bigvee\{\overline{\Gamma} \mid A \in \Gamma$ and $\Gamma \subseteq \mathcal{K}\}$ holds.*

This property is inherited from the freely generated boolean algebra $\mathcal{B}_X$ we introduced in the preceding proof. We now associate to a finite set $\mathcal{K}$ of formulae of $\mathcal{L}$, a finite set $\psi(\mathcal{K})$ containing formulae of potentially greater rank:
$\psi(\mathcal{K}) = \{\overline{\Gamma} * \overline{\Delta} \mid \Gamma, \Delta \in \mathcal{P}(\mathcal{K})\} \cup \{\neg(\overline{\Gamma} \mathrel{-\!\!*} \neg\overline{\Delta}) \mid \Gamma, \Delta \in \mathcal{P}(\mathcal{K})\}$

Proposition 11. *If $\mathcal{K}$ is a finite subset of $\mathcal{L}_r$ then $\psi(\mathcal{K})$ is a finite subset of $\mathcal{L}_{r+1}$. If $\mathsf{I} \in \mathcal{K}$ then $\mathcal{K} \subseteq \mathcal{B}(\psi(\mathcal{K}))$.*

Proof. The first result is trivial. If $\mathsf{I} \in \mathcal{K}$ then $\mathsf{I} = \bigvee\{\overline{\Gamma} \mid \mathsf{I} \in \Gamma$ and $\Gamma \in \mathcal{P}(\mathcal{K})\}$ by Proposition 10. Let $A \in \mathcal{K}$, by Proposition 10, we have $A = \bigvee\{\overline{\Delta} \mid A \in \Delta$ and $\Delta \in \mathcal{P}(\mathcal{K})\}$. Then, by distributivity of $*$ over $\vee$ (see Proposition 1), we obtain the identities $A = \mathsf{I} * A = \bigvee\{\overline{\Gamma} * \overline{\Delta} \mid \mathsf{I} \in \Gamma, A \in \Delta$ and $\Gamma, \Delta \in \mathcal{P}(\mathcal{K})\}$. Then $A \in \mathcal{B}(\psi(\mathcal{K}))$.

Proposition 12. *If $\mathcal{L}_r$ is finite then $\mathcal{L}_{r+1} = \mathcal{B}(\psi(\mathcal{L}_r))$.*

Theorem 6. Var *is finite iff $\mathcal{L}_0$ is finite iff for all r, $\mathcal{L}_r$ is finite.*

5.2 Observational Equivalence and Bisimulation

Now we use formulae of BBI and the forcing relation to distinguish between elements of relational frames. We suppose that the set of propositional variables Var is finite and consider a fixed relational frame $(\mathcal{M}, \triangleright, \mathsf{e})$. We also have a fixed interpretation $v(X) \subseteq \mathcal{M}$ for each propositional variable X.

The valuation v is the atomic observational tool to distinguish between elements of $\mathcal{M}$. X distinguishes the elements of $v(X)$ from the elements of $\mathcal{M} - v(X)$ and I distinguishes e from the other elements of $\mathcal{M}$. We define the *atomic observational equivalence* $\sim_0$ by $a \sim_0 b$ if $\forall F \in \mathsf{Var} \cup \{\mathsf{I}\}, a \Vdash F$ iff $b \Vdash F$. So $a \sim_0 b$ holds when no atomic observation can distinguish a from b. Then we generalize the observational equivalence to a subset $\mathcal{K}$ of $\mathcal{L}$. We define $\sim_{\mathcal{K}}$, the *observational equivalence under* $\mathcal{K}$ by: $a \sim_{\mathcal{K}} b$ iff $\forall F \in \mathcal{K}, a \Vdash F$ iff $b \Vdash F$. Then a and b are observationally equivalent under $\mathcal{K}$ when they cannot be distinguished from each other using forcing and formulae of $\mathcal{K}$. Then they are neither distinguishable by any boolean combination of formulae of K.

Proposition 13. $\sim_{\mathcal{K}} = \sim_{\mathcal{B}(\mathcal{K})}$.

Now we suppose that $\mathcal{K}$ is a finite subset of $\mathcal{L}$. Given a in $\mathcal{M}$, we define the subset $\mathcal{K}_a$ of $\mathcal{K}$ by $\mathcal{K}_a = \{F \in \mathcal{K} \mid a \Vdash F\}$. $\overline{\mathcal{K}_a}$ characterizes the $\sim_{\mathcal{K}}$-class of a.

Proposition 14. *For any $a, b \in \mathcal{M}$, $a \sim_{\mathcal{K}} b$ if and only if $b \Vdash \overline{\mathcal{K}_a}$.*

Definition 6. *We define $\sim_\omega$, the* observational equivalence *by $\sim_\omega = \sim_{\mathcal{L}}$ and the* observational equivalence up to rank r *by $\sim_r = \sim_{\mathcal{L}_r}$.*

This definition is coherent with the previous definition of $\sim_0$ because of $\mathcal{L}_0 = \mathcal{B}(\mathsf{Var} \cup \{\mathsf{I}\})$ and Proposition 13: the atomic observational equivalence coincides with the observational equivalence up to rank 0. We now generalize this identity for rank r. We recall the notion of bisimulation. We define an increasing operator $\mathcal{F} : \mathcal{P}(\mathcal{M}^2) \longrightarrow \mathcal{P}(\mathcal{M}^2)$ on the set of binary relation over $\mathcal{M}$. Let $R \in \mathcal{P}(\mathcal{M}^2)$ be a binary relation on $\mathcal{M}$. Then $\mathcal{F}(R)$ is the binary relation on $\mathcal{M}$ characterized by:

$$m\ \mathcal{F}(R)\ m' \quad \text{iff} \quad \begin{cases} \forall a, b \rhd m,\ \exists a', b' \rhd m',\ a\,R\,a' \text{ and } b\,R\,b' \\ \forall a', b' \rhd m',\ \exists a, b \rhd m,\ a\,R\,a' \text{ and } b\,R\,b' \\ \forall m, a \rhd b,\ \exists m', a' \rhd b',\ a\,R\,a' \text{ and } b\,R\,b' \\ \forall m', a' \rhd b',\ \exists m, a \rhd b,\ a\,R\,a' \text{ and } b\,R\,b' \end{cases}$$

With this definition, we could check that the full relation $\mathcal{M}^2$ is a fixpoint of $\mathcal{F}$, i.e. $\mathcal{F}(\mathcal{M}^2) = \mathcal{M}^2$. In order to obtain the bisimulation, we combine $\mathcal{F}$ with an atomic distinction feature using the $\sim_0$ atomic observational equivalence.

Definition 7. *The* bisimulation equivalence $\sim$ *is the greatest fixpoint of the increasing function $\mathcal{F}_0$ where $\mathcal{F}_0(R) = \mathcal{F}(R) \cap \sim_0$.*

As noted by Milner [10], $\sim$ could be obtained either by the union of all bisimulations (i.e. binary relations satisfying $R \subseteq \mathcal{F}_0(R)$) or as the limit of the decreasing *transfinite* sequence $\bigcap_\lambda \mathcal{F}_0^\lambda(\mathcal{M}^2)$, λ ranges over the class of *ordinals*.

5.3 The Observational Power of BBI

The function $\mathcal{F}$ operates on binary relations and thus on observational equivalences $\sim_{\mathcal{K}}$. The next result shows when $\mathcal{K}$ is finite, the behavior of $\mathcal{F}$ on $\sim_{\mathcal{K}}$ can be represented by a finitary transformation on the set $\mathcal{K}$.

Lemma 3. *For any finite subset $\mathcal{K}$ of $\mathcal{L}$, $\mathcal{F}(\sim_{\mathcal{K}}) = \sim_{\psi(\mathcal{K})}$.*

Theorem 7. *For any rank r, $\mathcal{F}(\sim_r) = \sim_{r+1}$ and $\sim_r = \mathcal{F}^r(\sim_0)$.*

Proof. Using Propositions 12, 13 and Lemma 3 we derive $\mathcal{F}(\sim_r) = \mathcal{F}(\sim_{\mathcal{L}_r}) = \sim_{\psi(\mathcal{L}_r)} = \sim_{\mathcal{B}(\psi(\mathcal{L}_r))} = \sim_{\mathcal{L}_{r+1}} = \sim_{r+1}$. Then by induction on r, we prove $\sim_r = \mathcal{F}^r(\sim_0)$.

Corollary 3. *Observational equivalence is the ω-limit of the decreasing sequence*

$$\mathcal{M}^2 \supseteq \mathcal{F}_0(\mathcal{M}^2) \supseteq \mathcal{F}_0^2(\mathcal{M}^2) \supseteq \cdots \supseteq \bigcap_{r<\omega} \mathcal{F}_0^r(\mathcal{M}^2) = \sim_\omega \supseteq \cdots \supseteq \bigcap_{\lambda} \mathcal{F}_0^\lambda(\mathcal{M}^2) = \sim$$

Proof. We prove $\mathcal{F}_0^{r+1}(\mathcal{M}^2) = \sim_r$ by induction on r once having noticed that $\mathcal{F}_0(\mathcal{M}^2) = \sim_0$. Then, any pre-fixpoint of $\mathcal{F}_0$ (i.e. any bisimulation, including $\sim$) is smaller than any element of the transfinite decreasing sequence $\mathcal{F}_0^\lambda(\mathcal{M}^2)$, and in particular when $\lambda = \omega$.

Observational equivalence $\sim_\omega$ is *not necessarily equal* to bisimulation equivalence $\sim$ because iterations up to ordinals λ greater than ω could be necessary to reach the greatest fixpoint $\bigcap_\lambda \mathcal{F}_0^\lambda(\mathcal{M}^2)$. As Milner noticed [10], one should use infinitary logics to make infinite observations. In this context, our results can be related to a recent study on resources and processes based on BBI [13] and provide a characterization of the observational power of BBI. Though in general $\sim$ is not equal to $\sim_\omega$, it is interesting to study under which further conditions the identity $\sim\, = \,\sim_\omega$ holds. For example, it holds when $\mathcal{M}$ is finite or when the relation $\triangleright$ is *locally finite* or more generally, when the model has the *Hennessy-Milner property*. The results obtained in the context of modal logic [1] could be adapted to BBI. To have $\sim\, = \,\sim_\omega$ is an important goal that could provide constructive tools to show equivalence and also to distinguish.

References

1. P. Blackburn, M. de Rijke, and Y. Venema. *Modal logic*. Cambridge University Press, New York, NY, USA, 2001.
2. L. Caires and L. Cardelli. A Spatial Logic for Concurrency (part II). In *Int. Conf. on Concurrency Theory, CONCUR 2002, LNCS 2421*, pages 209–225, 2002.
3. L. Caires and E. Lozes. Elimination of quantifiers and undecidability in spatial logics for concurrency. In *CONCUR 2004, LNCS 3170*, pages 240–257, 2004.
4. B. Davey and H. Priestley. *Introduction to Lattices and Order*. Cambridge Mathematical Textbooks. Cambridge University Press, 1990.
5. A. Dawar, P. Gardner, and G. Ghelli. Adjunct Elimination through Games in Static Ambient Logic. In *FSTTCS 2004, LNCS 3328*, pages 211–223, 2004.
6. D. Galmiche and D. Méry. Semantic labelled tableaux for propositional BI without bottom. *Journal of Logic and Computation*, 13(5):707–753, 2003.
7. D. Galmiche and D. Méry. Characterizing provability in BI's pointer logic through resource graphs. In *LPAR 2005, LNAI 3835*, pages 459–473, Montego Bay, Jamaica, December 2005.
8. D. Galmiche, D. Méry, and D.Pym. The semantics of BI and Resource Tableaux. *Math. Struct. in Comp. Science*, 15(6):1033–1088, 2005.
9. S. Ishtiaq and P. O'Hearn. BI as an assertion language for mutable data structures. In *28th ACM Symposium on Principles of Programming Languages*, pages 14–26, London, 2001.
10. R. Milner. *Communication and Concurrency*. International series in computer science. Prentice Hall, 1989.
11. P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *15th Int. Work. on Computer Science Logic, LNCS 2142*, pages 1–19, Paris, 2001.
12. D. Pym. *The Semantics and Proof Theory of the Logic of Bunched Implications*, volume 26 of *Applied Logic Series*. Kluwer Academic Publishers, 2002.
13. D. Pym and C. Tofts. A calculus and logic of resources and processes. Technical Report HPL-2004-170, HP Labs, 2004.
14. J. Reynolds. Separation logic: A logic for shared mutable data structures. In *IEEE Symposium on Logic in Computer Science*, pages 55–74, Copenhagen, July 2002.
15. R. Statman. Intuitionistic Propositional Logic is Polynomial-Space Complete. *Theoretical Computer Science*, 9:67–72, 1979.
16. H. Yang. Ternary-relation Models of Boolean BI: Soundness and Completeness. Unpublished note, 2004.

On Continuous Timed Automata with Input-Determined Guards

Fabrice Chevalier[1], Deepak D'Souza[2], and Pavithra Prabhakar[2]

[1] LSV, ENS de Cachan
61 Av. Pres. Wilson, Cachan Cedex 94235, France
fabrice.chevalier@lsv.ens-cachan.fr
[2] Department of Computer Science and Automation
Indian Institute of Science, Bangalore 560012, India
{deepakd, pavithra}@csa.iisc.ernet.in

Abstract. We consider a general class of timed automata parameterized by a set of "input-determined" operators, in a continuous time setting. We show that for any such set of operators, we have a monadic second order logic characterization of the class of timed languages accepted by the corresponding class of automata. Further, we consider natural timed temporal logics based on these operators, and show that they are expressively equivalent to the first-order fragment of the corresponding MSO logics. As a corollary of these general results we obtain an expressive completeness result for the continuous version of MTL.

1 Introduction

Timed automata are a popular model of real-time systems, introduced by Alur and Dill in the early nineties [1]. Since then there have been several variants of these automata based on *input-determined* guards [2,3,4,5]. Unlike the explicit clock based guards of timed automata, an input-determined guard is based on a distance operator whose value is completely determined by the input timed word and a time point in it. This property leads to robust logical properties including closure under complementation which timed automata lack. A good example of an input-determined operator is the event-recording operator $\lhd_a$ of [2] which measures the distance to the last time an event a occurred. Similarly the "eventual" operator $\Diamond_a$ [6,5] inspired by the well-known timed logic Metric Temporal Logic (MTL) [7,2,8], measures the time to "some" future occurrence of an a event.

There have been two natural ways of employing these operators in automata and logical formalisms in the literature. One is the traditional "pointwise" interpretation in which guards are asserted only at "action-points" in a timed word. The other is the so-called "continuous" interpretation in which assertions can be made at *any* time point along the timed word. The two interpretations are well illustrated by the MTL formula $\Diamond\Diamond_{[1,1]}a$ which states that there is a point in future such that an a occurs exactly one time unit later. In the pointwise semantics, the formula is not satisfied by the timed word which comprises a b at time 1 followed by an a at time 3, but is satisfied in the continuous semantics. In general, the continuous semantics is strictly more expressive than the pointwise semantics [9,10].

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 369–380, 2006.

In the pointwise semantics, the work in [6] provides a general framework for showing determinizability, closure properties, and monadic second-order (MSO) logic characterizations, for classes of timed automata based on input-determined operators, called input-determined automata (IDA's). It also identifies natural timed temporal logics based on these operators which are expressively complete with respect to the corresponding automata classes.

In this paper we show a similar general framework for the continuous semantics. Thus we first define an appropriate "continuous" version of these automata called continuous input-determined automata (CIDA's) which are parameterized by a set of input-determined operators. These CIDA's extend IDA's by allowing epsilon-transitions and state invariants. We show that these classes of automata are determinizable and closed under boolean operations. They also admit logical characterizations via natural MSO logics based on the input-determined operators, and interpreted over continuous time. Further, the continuous version of the natural timed temporal logics based on these operators are shown to be expressively complete, in that they correspond to the first-order fragments of the associated MSO logics. These results generalize to the corresponding *recursive* formalisms where the input-determined operators take as arguments logical formulas or "floating" automata, as originally used in the work of [11].

This framework can be used as a general technique for showing such results for any class of automata and logics based on input-determined operators. In particular, the results of [11] for the class of recursive event clock automata (ECA's), pertaining to the MSO characterization via the logic MinMaxML and the expressive completeness of recursive Event Clock Temporal Logic (ECTL), follow as corollaries of our results.

As a new application, we obtain an expressive completeness result for MTL in the continuous semantics. MTL can be viewed as the recursive timed temporal logic based on the operator $\Diamond$, and hence corresponds to the first-order fragment of recursive CIDA's and the MSO based on the operator $\Diamond$.

The techniques used to prove our results are similar to [6] in that we also make use of the notion of *proper* alphabets. These alphabets help in determinizing CIDA's and showing closure properties. For the MSO characterization we use proper alphabets to translate formulas into a continuous version of Büchi's MSO logic, which preserves, in a sense, the original models of the formula. Now we need to make use of the fact that the "untiming" of continuous MSO formulas is regular in order to obtain a CIDA for the original MSO formula. We give an automata-theoretic proof of this result which was independently proved by Rabinovich in [12] using a translation to classical MSO. For the expressive completeness result concerning our timed temporal logics we factor through the well-known result of Kamp for classical LTL [13].

The technique used in [11,14] for event clock automata is similar in that they factor through Kamp's theorem to prove their expressive completeness result. However the MSO characterization is obtained differently by showing that quantified ECTL is expressively equivalent to recursive ECA's.

In this paper we deal with finite timed words, though the results can be easily extended to infinite words as well. Details of proofs omitted due to lack of space can be found in the technical report [15].

2 Preliminaries

For an alphabet A, we use A^* to denote the set of finite words over A. For a word w in A^*, we use $|w|$ to denote its length. We make use of the standard notations for regular expressions, with '$\cdot$' for concatenation and '*' for Kleene closure.

A *finite state automaton (FSA)* $\mathcal{A}$ over a finite alphabet A is a structure $\mathcal{A} = (Q, s, \delta, F)$, where Q is a finite set of states, s is the initial state, $\delta \subseteq Q \times A \times Q$ is the set of transitions, and $F \subseteq Q$ is the set of final states. A run ρ of $\mathcal{A}$ on a word $w = a_1 \cdots a_n \in A^*$ is a mapping from $\{0, \cdots, n\} \rightarrow Q$ such that $(\rho(i), a_{i+1}, \rho(i+1)) \in \delta$ for each $i < n$, and $\rho(0) = s$. The run is accepting if $\rho(n) \in F$. The symbolic language accepted by $\mathcal{A}$, denoted $L_{sym}(\mathcal{A})$, is the set of words in A^* over which $\mathcal{A}$ has an accepting run.

We denote the set of non-negative and positive real numbers by $\mathbb{R}_{\geq 0}$. We use $\mathcal{I}_{\mathbb{R}_{\geq 0}}$ to denote the set of intervals, where an interval is a convex subset of $\mathbb{R}_{\geq 0}$. Two interval I and J are *adjacent* if $I \cap J = \emptyset$ and $I \cup J$ is an interval. We use $\mathcal{I}_{\mathbb{Q}}$ to denote the set of intervals whose end-points are rational or ∞.

Let A be an alphabet and let $f : [0, r] \rightarrow A$ be a function, where $r \in \mathbb{R}_{\geq 0}$. We denote r by *length*(f). We call f a *finitely varying* function over A, if there exist a word $a_0 a_1 \cdots a_{2n}$ in A^*, and an interval sequence $I_0 I_1 \cdots I_{2n}$, such that $0 \in I_0$, I_i and I_{i+1} are adjacent for each i, I_i is singular if i is even, and for all $t \in [0, r]$, $f(t) = a_i$ if $t \in I_i$. We then call $(a_0, I_0) \cdots (a_{2n}, I_{2n})$ an *interval representation* of f. We call a word $a_0 a_1 \cdots a_n$ in A^* *canonical*, if n is even, and there does not exist an even i such that $0 < i < n$ and $a_{i-1} = a_i = a_{i+1}$. An interval representation $(b_0, I_0) \cdots (b_{2n}, I_{2n})$ of f is called *canonical*, if $b_0 \cdots b_{2n}$ is canonical. Note that every finitely varying function has a canonical interval representation. We define *func*(A) to be the set of all finitely varying functions over A.

Let $f \in$ *func*(A) and let $(a_0, I_0) \cdots (a_{2n}, I_{2n})$ be its canonical interval representation. We denote the untiming of the function as a sequence which captures explicitly the points of discontinuities and the intervals between them. The untiming of the above f, denoted $untiming(f)$, is defined as $a_0 \cdots a_{2n}$. Note that the untiming of a function is always canonical. Given a word w in A^*, we define its timing to be a set of functions: *timing*$(w) = \emptyset$ if $|w|$ is even, otherwise $f \in$ *timing*(w) if $w = a_0 a_1 \cdots a_{2n}$ and $(a_0, I_0)(a_1, I_1) \cdots (a_{2n}, I_{2n})$ is an interval representation of f. We can extend the definitions of *timing* and *untiming* to languages of functions in the expected way.

We define a timed word σ over an alphabet Σ to be an element of $(\Sigma \times \mathbb{R}_{\geq 0})^*$, such that $\sigma = (a_0, t_0)(a_1, t_1) \cdots (a_n, t_n)$ and $t_0 < t_1 < \cdots < t_n$. We denote the set of all timed words over Σ by $T\Sigma^*$. We define an *input-determined operator* Δ over an alphabet Σ as a partial function from $(T\Sigma^* \times \mathbb{R}_{\geq 0})$ to $2^{\mathbb{R}_{\geq 0}}$, which is defined for all pairs (σ, t), where $t \in [0, length(\sigma)]$. Given a set of input-determined operators Op, we define the set of guards over Op, denoted by $\mathcal{G}(Op)$, inductively as $g ::= \top \,|\, \Delta^I \,|\, \neg g \,|\, g \vee g \,|\, g \wedge g$, where $\Delta \in Op$ and $I \in \mathcal{I}_{\mathbb{Q}}$. Guards of the form Δ^I are called atomic. Given a timed word σ, we define the satisfiability of a guard g at time $t \in [0, length(\sigma)]$, denoted $\sigma, t \models g$, as $\sigma, t \models \Delta^I$ iff $\Delta(\sigma, t) \cap I \neq \emptyset$, and in the usual way for the boolean operators. For example $\Delta_{\mathbb{Q}}$, which maps (σ, t) to $\{1\}$ if t is rational and to $\{0\}$ otherwise, is an input-determined operator. Other examples include the eventual operator $\Diamond_a$, inspired by MTL, which maps (σ, t) to the set of time points in σ after t at which an event a occurs, and the event-recording operator $\lhd_a$ which maps (σ, t) to the set containing the time point which corresponds to the last occurrence of the event a before time t.

We call an input-determined operator Δ over Σ *finitely varying* if for all $\sigma \in T\Sigma^*$ and $I \in \mathcal{I}_{\mathbb{Q}}$, the function $f_\Delta : [0, \mathit{length}(\sigma)] \rightarrow \{0,1\}$ defined as, $f_\Delta(t)$ is 1 if $\sigma, t \models \Delta^I$, and 0 otherwise, is finitely varying. The operators $\Diamond_a$ and $\lhd_a$ are finitely varying, whereas $\Delta_{\mathbb{Q}}$ is not.

Let Σ be an alphabet and Op be a set of input determined operators over Σ. We call (Γ_1, Γ_2) a *symbolic alphabet* over (Σ, Op), if Γ_1 is a finite subset of $(\Sigma \cup \{\epsilon\}) \times \mathcal{G}(Op)$ and Γ_2 is a finite subset of $\mathcal{G}(Op)$. We define the set of timed words over Σ associated with a function f in $\mathit{func}(\Gamma_1 \cup \Gamma_2)$, denoted $tw(f)$, as follows. If $\mathit{untiming}(f) \notin \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$, then $tw(f) = \emptyset$. Otherwise, a timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n)$ is in $tw(f)$, provided for all $t \in [0, \mathit{length}(f)]$,

- $f(t) = (a, g)$, for some $a \in \Sigma$ and $g \in \mathcal{G}(Op)$, if there exists i such that $i \in \{1, \cdots, n\}$, $t_i = t$ and $a_i = a$, and if $\sigma, t \models g$, and
- $f(t) = (\epsilon, g)$ or g, for some $g \in \mathcal{G}(Op)$, if there does not exist i such that $i \in \{1, \cdots, n\}$, $t_i = t$, and if $\sigma, t \models g$.

Note that for any f, $tw(f)$ is either a singleton set or an empty set. We can extend the definition of tw to a set of functions as the union of the timed words corresponding to each function in the set.

Let G be a finite set of atomic guards over Op. We call (Γ_1, Γ_2) the *proper* symbolic alphabet over (Σ, Op) based on G, if $\Gamma_1 = (\Sigma \cup \{\epsilon\}) \times 2^G$ and $\Gamma_2 = 2^G$. A proper word is a word over a proper symbolic alphabet. Further we call a proper word γ over $(\Gamma_1 \cup \Gamma_2)$ *fully canonical*, if $\gamma \in \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ and no subword of γ is of the form $g \cdot (\epsilon, g) \cdot g$. If $f \in \mathit{func}(\Gamma_1 \cup \Gamma_2)$, then we associate with it the set of timed words obtained by interpreting $g \subseteq G$ as the guard $\bigwedge_{h \in g} h \wedge \bigwedge_{h \in G - g} \neg h$.

Example 1. Let $\Sigma = \{a\}$, $Op = \{\Diamond_a\}$ and $G = \{\Diamond_a^{[1,1]}\}$. The proper alphabet (Γ_1, Γ_2) is given by $\Gamma_1 = (\Sigma \cup \{\epsilon\}) \times 2^G$ and $\Gamma_2 = 2^G$. Let $f_1 : [0,2] \rightarrow \Gamma_1 \cup \Gamma_2$ such that $f_1(0) = f_1(2) = (a, \emptyset)$, $f_1(1) = (\epsilon, \{\Diamond_a^{[1,1]}\})$ and $f(t) = \emptyset$ if $t \neq 0, 1, 2$. We then have $tw(f_1) = \{(a,0)(a,2)\}$. Let $f_2 : [0,2] \rightarrow \Gamma_1 \cup \Gamma_2$ defined by $f_2(1) = (\epsilon, \emptyset)$ and $f_2(t) = f_1(t)$ if $t \neq 1$. Then $tw(f_2) = \emptyset$.

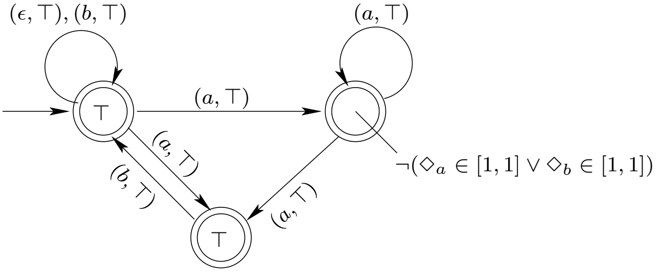
3 Continuous Input Determined Automata

Let Σ be an alphabet and Op be a set of input determined operators based on Σ. A *Continuous Input Determined Automaton* (*CIDA*) $\mathcal{A}$ over (Σ, Op) is a structure $(Q, s, \delta, F, \mathit{inv})$ on a symbolic alphabet (Γ_1, Γ_2) over (Σ, Op), where Q is a finite set of states, $s \in Q$ is the start state, $\delta \subseteq Q \times \Gamma_1 \times Q$ is the transition relation, $\mathit{inv} : Q \rightarrow \Gamma_2$ is the labelling function for the states, and $F \subseteq Q$ is the set of accepting states.

We now define the *symbolic language* accepted by the *CIDA* $\mathcal{A}$. Let $\gamma \in \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ and let $\gamma = \gamma_0 \gamma_1 \cdots \gamma_{2n}$. Let $N = \{0, \cdots, n+1\}$. A run of $\mathcal{A}$ over γ is a map $\rho : N \rightarrow Q$ such that $\rho(0) = s$, $(\rho(i), \gamma_{2i}, \rho(i+1)) \in \delta$ for $i = 0, \cdots, n$ and $\mathit{inv}(\rho(i)) = \gamma_{2i-1}$ for all $1 \leq i \leq n$. We say ρ is accepting if $\rho(n+1) \in F$. The symbolic language defined by $\mathcal{A}$, denoted $L_{sym}(\mathcal{A})$, is the set of words in $\Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ over which $\mathcal{A}$ has an accepting run. Note that a language L is a regular subset of $\Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ iff it is the symbolic language of a *CIDA*.

We define the language of functions accepted by the *CIDA* $\mathcal{A}$, denoted $F(\mathcal{A})$, as $timing(L_{sym}(\mathcal{A}))$. The timed language of the *CIDA* $\mathcal{A}$, denoted $L(\mathcal{A})$, is defined as $tw(F(\mathcal{A}))$.

We give below a concrete example of a *CIDA*, which we call *Continuous Eventual Timed Automata* (*CETA*). A *CETA* over an alphabet Σ is a *CIDA* over (Σ, Op), where $Op = \{\Diamond_a \mid a \in \Sigma\}$ is the set of eventual operators based on Σ. The diagram below gives a *CETA* over $\{a, b\}$ which recognizes the language L_{ni} (for "no insertion"), which consists of timed words in which between any two consecutive a's, there does not exist a time point from which at time distance one in the future there is an a or a b.


$(\epsilon, \top), (b, \top)$
$(a, \top)$
$(a, \top)$
$\top$
$(a, \top)$
$(b, \top)$
$\neg(\Diamond_a \in [1,1] \vee \Diamond_b \in [1,1])$
$(a, \top)$
$\top$

We define a *proper CIDA* to be a structure similar to *CIDA* except that it is over a proper symbolic alphabet instead of a symbolic alphabet. We call a proper *CIDA fully canonical* if its symbolic language consists of fully canonical proper words. We show below the closure of *CIDA*'s under the boolean operations. Let Σ be an alphabet and *Op* be a set of finitely varying operators.

Lemma 1. *CIDA's over* (Σ, Op) *and fully canonical proper CIDA's over* (Σ, Op) *define the same class of timed languages.*

Theorem 1. *The class of CIDA's over* (Σ, Op) *is closed under union, intersection and complementation.*

Proof. Union of *CIDA*'s is equivalent to the union of their symbolic languages. For complementation, using lemma 1 we can give an equivalent fully canonical proper *CIDA* $\mathcal{A}'$ for a given *CIDA* $\mathcal{A}$. But the set of timed words associated with two distinct fully canonical proper words is disjoint. Hence we can complement the timed language of $\mathcal{A}'$ by complementing its symbolic language with respect to the set of fully canonical proper words. □

4 Continuous Monadic Second Order Logic

In this section, we interpret Buchi's monadic second order logic over finitely varying functions and show that the untiming of the language of functions definable in the logic is regular.

Recall that for an alphabet A, B$\ddot{u}$chi's monadic second order logic (denoted here by $\mathrm{MSO}^c(A)$) is given as follows: $\varphi ::= Q_a(x) \mid x \in X \mid x < y \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x\varphi \mid \exists X\varphi$, where $a \in A$, and x and X are first and second order variables, respectively. We use the convention that the small letters are first order variables and capital letters are second order variables.

We interpret a formula of the logic over a finitely varying function f in $\mathit{func}(A)$, along with an interpretation $\mathbb{I}$ with respect to f, which assigns to a first order variable x, a value in $[0, \mathit{length}(f)]$, and to a set variable X, a finite subset of $[0, \mathit{length}(f)]$. We use $X \subseteq_{\mathit{fin}} Y$ to denote that X is a finite subset of Y.

For an interpretation $\mathbb{I}$, we use the notation $\mathbb{I}[t/x]$ to denote the interpretation which sends x to t and agrees with $\mathbb{I}$ on all other variables. Similarly, $\mathbb{I}[B/X]$ denotes the modification of $\mathbb{I}$ which maps the set variable X to B and the rest to the same as that by $\mathbb{I}$. We also use the notation $[t/x]$ to denote an interpretation which sends x to t when the rest of the interpretation is irrelevant.

We now define the semantics of $\mathrm{MSO}^c(A)$. Given a formula $\varphi \in \mathrm{MSO}^c(A)$, $f \in \mathit{func}(A)$ and an interpretation $\mathbb{I}$ with respect to f to the variables in φ, the satisfaction relation $f, \mathbb{I} \models \varphi$, is defined inductively as:

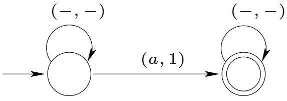
$$\begin{array}{ll}
f, \mathbb{I} \models \mathrm{Q}_a(x) & \text{iff } f(\mathbb{I}(x)) = a, \text{ where } a \in A. \\
f, \mathbb{I} \models x \in X & \text{iff } \mathbb{I}(x) \in \mathbb{I}(X). \\
f, \mathbb{I} \models x < y & \text{iff } \mathbb{I}(x) < \mathbb{I}(y). \\
f, \mathbb{I} \models \neg\varphi & \text{iff } f, \mathbb{I} \not\models \varphi. \\
f, \mathbb{I} \models \varphi_1 \vee \varphi_2 & \text{iff } f, \mathbb{I} \models \varphi_1 \text{ or } f, \mathbb{I} \models \varphi_2. \\
f, \mathbb{I} \models \exists x\varphi & \text{iff } \exists t \in [0, \mathit{length}(f)] : f, \mathbb{I}[t/x] \models \varphi. \\
f, \mathbb{I} \models \exists X\varphi & \text{iff } \exists B \subseteq_{\mathit{fin}} [0, \mathit{length}(f)] : f, \mathbb{I}[B/X] \models \varphi.
\end{array}$$

For a sentence, a formula without free variables, the interpretation does not play any role. Hence, for a sentence φ in $\mathrm{MSO}^c(A)$, we set the language defined by φ to be $F(\varphi) = \{f \in \mathit{func}(A) \mid f \models \varphi\}$. The following theorem relates FSA's and MSO^c.

Theorem 2. *Given a sentence φ in $\mathrm{MSO}^c(A)$, we can give a finite state automaton $\mathcal{A}_\varphi$ such that $F(\varphi) = \mathit{timing}(L_{sym}(\mathcal{A}_\varphi))$.*

Proof. We construct the automaton for a formula $\varphi \in \mathrm{MSO}^c(A)$, inductively. Let $X = (x_1, x_2, \cdots, x_n)$ and $Y = (X_1, X_2, \cdots, X_m)$ be the free variables in φ. We give an automaton $\mathcal{A}_\varphi^{X,Y}$ over $A' = A \times \{0,1\}^{n+m}$, which is related to φ as follows. Let $f \in \mathit{func}(A)$ and $\mathbb{I}$ be an interpretation of the variables in (X, Y) with respect to f. Then $f, \mathbb{I} \models \varphi$ iff $\mathit{untiming}(f_{\mathbb{I}}^{(X,Y)}) \in \mathcal{A}_\varphi^{X,Y}$. The function $f_{\mathbb{I}}^{(X,Y)} : [0, \mathit{length}(f)] \to A'$ is defined as, $f_{\mathbb{I}}^{(X,Y)}(t) = (f(t), i_1, i_2, \cdots, i_n, j_1, j_2, \cdots, j_m)$, where $i_k = 1$ if $\mathbb{I}(x_k) = t$ and 0 otherwise, and $j_k = 1$ if $t \in \mathbb{I}(X_k)$ and 0 otherwise. Let $\mathcal{A}_{canon}^i$ be the automaton which accepts canonical words over $A \times \{0,1\}^i$. We consider here the cases when φ is $Q_a(x)$ and $\exists x\varphi$, and the detailed proof can be found in [15].

If $\varphi = Q_a(x)$, then the automaton $\mathcal{A}_\varphi^{X,Y}$ is the intersection of $\mathcal{A}_{canon}^1$ with:



Suppose $\varphi = \exists x\eta$. Let $\mathcal{A}_{\eta}^{X,Y}$ be the automaton for η, where (X, Y) are the free variables in η, and $X = (x, x_1, \cdots, x_n)$. Let $X' = (x_1, \cdots, x_n)$. We first intersect $\mathcal{A}_{\eta}^{X,Y}$ with $\mathcal{A}_{valid}$, which accepts words in which there is exactly one symbol with a 1 for its x-component at some even position (assuming indices start from 0). We then project away the x-components of the labels on the transitions in the automaton. Next we canonicalize the resulting automaton in two steps. First we convert the automaton to one that is in the form of a bipartite graph in which the transitions are only from the states in one set to the other. We then add transitions as described below repeatedly until no more can be added. A transition (p, a, r) is added if there exist transitions (p, a, q), (q, a, q') and (q', a, r). The above construction relies on the fact that if $f_{\mathbb{I}}^{(X,Y)}$ is in the timing of w, then $f_{\mathbb{I}'}^{(X',Y)}$ is in the timing of w', where w' is obtained from w by projecting away its x-component and $\mathbb{I}'$ is an interpretation to the variables in (X', Y) which agrees with $\mathbb{I}$ on the common variables. Finally we intersect the automaton with $\mathcal{A}_{canon}^{n+m}$ where m is the number of variables in Y. □

5 A Logical Characterization of *CIDA*'s

In this section we give a logical characterization of *CIDA*'s in terms of a monadic second order logic parameterized by a set of input-determined operators. Let Σ be an alphabet and *Op* be a set of input determined operators over Σ. We define the syntax of *continuous timed monadic second order logic* over (Σ, Op)(TMSO$^c(\Sigma, Op)$) as:

$$\varphi ::= Q_a(x) \mid \Delta^I(x) \mid x \in X \mid x < y \mid \neg\varphi \mid (\varphi \vee \varphi) \mid \exists x\varphi \mid \exists X\varphi,$$

where $a \in \Sigma$, $\Delta \in Op$, $I \in \mathcal{I}_{\mathbb{Q}}$, and x and X are first and second order variables. We interpret the logic over timed words in $T\Sigma^*$. Given a formula $\varphi \in$ TMSO$^c(\Sigma, Op)$, a timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n)$ in $T\Sigma^*$, and an interpretation $\mathbb{I}$ with respect to σ, which maps a first order variable x to $t \in [0, \mathit{length}(\sigma)]$ and a second order variable X to $B \subseteq_{fin} [0, \mathit{length}(\sigma)]$, we define the satisfaction relation $\sigma, \mathbb{I} \models \mathrm{Q}_a(x)$ as $\exists i : a_i = a, t_i = \mathbb{I}(x)$, and $\sigma, \mathbb{I} \models \Delta^I(x)$ as $\Delta(\sigma, \mathbb{I}(x)) \cap I \neq \emptyset$, and the rest of the cases are similar to that of MSO^c over functions. For a sentence φ in TMSO$^c(\Sigma, Op)$, we set the timed language defined by φ to be $L(\varphi) = \{\sigma \in \mathrm{T}\Sigma^* \mid \sigma \models \varphi\}$. We now show that TMSO^c characterizes *CIDA*'s.

Theorem 3. *Let Σ be a finite alphabet and Op be a set of finitely varying input-determined operators based on Σ. Let L be a timed language over Σ. Then L is accepted by a CIDA over (Σ, Op) iff it is definable by a* TMSO$^c(\Sigma, Op)$ *sentence.*

We devote the rest of the section for a proof of the above theorem. As a proof of the forward direction, we show that the class of languages defined by proper *CIDA*'s over (Σ, Op) is a subset of the class of languages defined by TMSO$^c(\Sigma, Op)$ sentences. Let $\mathcal{A} = (Q, s, \delta, F, inv)$ be a proper *CIDA* over (Γ_1, Γ_2) based on a set of atomic guards G over Op. We give a formula $\varphi_{\mathcal{A}}$ such that $L(\mathcal{A}) = L(\varphi_{\mathcal{A}})$. The formula essentially checks for the existence of a valid run of $\mathcal{A}$ over the timed words. Let $\delta = \{e_1, \cdots, e_m\}$ be the set of transitions. We set $(e, e') \in \mathit{consec}$ if and only if there exists q such that

$e = (p, \gamma, q)$ and $e' = (q, \gamma', r)$. We use $action(x)$ for $\bigvee_{a \in \Sigma} Q_a(x)$. Given $g \subseteq G$, we will use $g(x)$ to denote the TMSO^c formula $\bigwedge_{\Delta^I \in g} \Delta^I(x) \wedge \bigwedge_{\Delta^I \in G-g} \neg\Delta^I(x)$. The second order variables $X_{e_1}, \cdots, X_{e_m}$ are used to capture the points in the timed words which correspond to the transitions $e_1, \cdots, e_m$, respectively, and X to capture their union. Let $between(x, y, z) = x < y \wedge y < z$, $first(x) = \neg\exists y(y < x)$, $last(x) = \neg\exists y(x < y)$ and $next(x, y, X) = x \in X \wedge y \in X \wedge \neg\exists w(x < w \wedge w < y \wedge w \in X)$.

$\varphi_{\mathcal{A}}$ is given by: $\exists X \exists X_{e_1} \cdots \exists X_{e_m}(\varphi_1 \wedge \varphi_2 \wedge \varphi_3 \wedge \varphi_4 \wedge \varphi_5 \wedge \varphi_6 \wedge \varphi_7)$, where:

$$\varphi_1 : \forall x(\bigvee_{e \in \delta} x \in X_e \Leftrightarrow x \in X) \bigwedge \forall x \bigwedge_{i,j \in \{1,\cdots,m\}, i \neq j}(x \in X_{e_i} \Rightarrow x \notin X_{e_j}).$$
$$\varphi_2 : \forall x(first(x) \Rightarrow \bigvee_{(s,\gamma,q) \in \delta} x \in X_{(s,\gamma,q)}).$$
$$\varphi_3 : \forall x(last(x) \Rightarrow \bigvee_{(q,\gamma,f) \in \delta, f \in F} x \in X_{(q,\gamma,f)}).$$
$$\varphi_4 : \forall x \forall y(next(x, y, X) \Rightarrow \bigvee_{e,e' \in consec}(x \in X_e \wedge y \in X_{e'})).$$
$$\varphi_5 : \forall x \bigwedge_{(p,(a,g),q) \in \delta}(x \in X_{(p,(a,g),q)} \Rightarrow (Q_a(x) \wedge g(x))).$$
$$\varphi_6 : \forall x \bigwedge_{(p,(\epsilon,g),q) \in \delta}(x \in X_{(p,(\epsilon,g),q)} \Rightarrow (\neg action(x) \wedge g(x))).$$
$$\varphi_7 : \forall x \forall y \forall z((next(y, z) \wedge between(y, x, z)) \Rightarrow$$
$$(\bigwedge_{(p,a,q) \in \delta}(y \in X_{(p,a,q)} \Rightarrow (\neg action(x) \wedge [inv(q)](x))))).$$

In the other direction we reduce a TMSO^c formula to an MSO^c formula and then factor through theorem 2 to get an FSA over $\Gamma_1 \cup \Gamma_2$. Let $\varphi \in$ TMSO$^c(\Sigma, Op)$ and let $G = \{\Delta^I \mid \Delta^I(x)$ is a subformula of $\varphi\}$. Let (Γ_1, Γ_2) be the proper alphabet over (Σ, Op) based on G, and let $\Gamma = \Gamma_1 \cup \Gamma_2$. We now give the function *tmso-mso*, which maps a TMSO$^c(\Sigma, Op)$ formula φ to the MSO$^c(\Gamma)$ formula obtained by replacing every atomic formula $Q_a(x)$ by $\bigvee_{(a,g) \in \Gamma} Q_{(a,g)}(x)$ and $\Delta^I(x)$ by $\bigvee_{(c,g) \in \Gamma, \Delta^I \in g} Q_{(c,g)}(x) \vee \bigvee_{g \in \Gamma, \Delta^I \in g} Q_g(x)$.

Theorem 4. *Given a sentence* $\varphi \in$ TMSO$^c(\Sigma, Op)$, $L(\varphi) = tw(F(tmso\text{-}mso\ (\varphi)))$.

$$\begin{tikzcd}
\mathrm{TMSO}^c - \varphi \arrow[r, dashed] \arrow[d] & \mathit{CIDA} - \mathcal{A}' \\
\mathrm{MSO}^c - \tilde{\varphi} \arrow[r] & \mathit{FSA} - \mathcal{A}_{\tilde{\varphi}} \arrow[u]
\end{tikzcd}$$

We can now complete the proof by taking the route in the diagram above. From theorem 4, $L(\varphi) = tw(F(\tilde{\varphi}))$, where $\tilde{\varphi} = tmso\text{-}mso(\varphi)$. By theorem 2 there exists an FSA $\mathcal{A}_{\tilde{\varphi}}$ such that $F(L_{sym}(\mathcal{A}_{\tilde{\varphi}})) = F(\tilde{\varphi})$. Hence $L(\varphi) = tw(F(L_{sym}(\mathcal{A}_{\tilde{\varphi}})))$. We can assume that $L_{sym}(\mathcal{A}_{\tilde{\varphi}}) \subseteq \Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ as words not in $\Gamma_1 \cdot (\Gamma_2 \cdot \Gamma_1)^*$ do not have any timed words associated with them. Thus we can give a $CIDA$ $\mathcal{A}'$ such that $L_{sym}(\mathcal{A}') = L_{sym}(\mathcal{A}_{\tilde{\varphi}})$. It now follows that $L(\varphi) = L(\mathcal{A}')$.

6 Continuous Timed Linear Temporal Logic

In this section we identify a natural, expressively complete, timed linear temporal logic based on a set of input-determined operators. The logic is denoted TLTL$^c(\Sigma, Op)$, parameterized by the alphabet Σ and the set of input-determined operators Op over Σ. The formulas of TLTL^c are given by:

$$\theta ::= a \mid \Delta^I \mid (\theta U \theta) \mid (\theta S \theta) \mid \neg\theta \mid (\theta \vee \theta),$$

where $a \in \Sigma$, $\Delta \in Op$ and $I \in \mathcal{I}_{\mathbb{Q}}$. We interpret $\mathrm{TLTL}^c(\Sigma, Op)$ formulas over timed words over Σ. Let φ be a $\mathrm{TLTL}^c(\Sigma, Op)$ formula. Let $\sigma \in \mathrm{T}\Sigma^*$, with $\sigma = (a_1, t_1) \cdots (a_n, t_n)$ and let $t \in [0, \mathit{length}(\sigma)]$. Then the satisfaction relation $\sigma, t \models \varphi$ is given by:

$\sigma, t \models a$ iff $\exists i : a_i = a, t_i = t$.
$\sigma, t \models \Delta^I$ iff $\Delta(\sigma, t) \cap I \neq \emptyset$.
$\sigma, t \models \theta U \eta$ iff $\exists t' : t < t' \leq \mathit{length}(\sigma), \sigma, t' \models \eta, \forall t'' : t < t'' < t', \sigma, t'' \models \theta$.
$\sigma, t \models \theta S \eta$ iff $\exists t' : 0 \leq t' < t, \sigma, t' \models \eta, \forall t'' : t' < t'' < t, \sigma, t'' \models \theta$.

It is defined in the usual manner for the boolean combinations. The language defined by a $\mathrm{TLTL}^c(\Sigma, Op)$ formula θ is given by $L(\theta) = \{\sigma \in T\Sigma^* \mid \sigma, 0 \models \theta\}$.

We show that TLTL^c is expressively equivalent to the first order fragment of TMSO^c. Let us denote by $\mathrm{TFO}^c(\Sigma, Op)$ the first order fragment of $\mathrm{TMSO}^c(\Sigma, Op)$ (i,e, the fragment we get by disallowing quantification over set variables). The logics TLTL^c and TFO^c are expressively equivalent in the following sense:

Theorem 5. *Let Σ be an alphabet and Op be a set of finitely varying input-determined operators over Σ. A timed language $L \subseteq T\Sigma^*$ is definable by a* $\mathrm{TLTL}^c(\Sigma, Op)$ *formula θ iff it is definable by a sentence φ in* $\mathrm{TFO}^c(\Sigma, Op)$.

Proof. The proof of the forward direction is similar to the classical translation of LTL to MSO. In the converse direction a more transparent proof is obtained by factoring through Kamp's result for classical LTL^c. Recall that the syntax of $\mathrm{LTL}^c(A)$ is given by: $\theta ::= a \mid (\theta U \theta) \mid (\theta S \theta) \mid \neg\theta \mid (\theta \vee \theta)$, where $a \in A$. The logic is interpreted over functions $f \in \mathit{func}(A)$. Given $t \in [0, \mathit{length}(f)]$ and $\theta \in \mathrm{LTL}^c(A)$, the satisfaction relation $f, t \models a$ is defined as $f(t) = a$, and for the rest of the cases it is defined as for TLTL^c. Let $\mathrm{FO}^c(A)$ denote the first order fragment of $\mathrm{MSO}^c(A)$. Then the result due to Kamp [13] states that:

Theorem 6 ([13]). $\mathrm{LTL}^c(A)$ *is expressively equivalent to* $\mathrm{FO}^c(A)$.

Let φ be a $\mathrm{TFO}^c(\Sigma, Op)$ sentence. By theorem 4 the function *tmso-mso* maps a $\mathrm{TFO}^c(\Sigma, Op)$ formula to an $\mathrm{FO}^c(\Gamma)$ formula $\tilde{\varphi} = \mathit{tmso\text{-}mso}(\varphi)$ such that $L(\varphi) = \mathit{tw}(F(\tilde{\varphi}))$. By Kamp's result, there exists a mapping *fo-ltl* such that $F(\tilde{\varphi}) = F(\mathit{fo\text{-}ltl}(\tilde{\varphi}))$. Let $\theta = \mathit{fo\text{-}ltl}(\tilde{\varphi})$. Let $a \in \Sigma$, $g \in G$ and $(a, g) \in \Gamma$. Let $\theta_g = \bigwedge_{h \in g} h \wedge \bigwedge_{h \in G-g} \neg h$. We define the function *ltl-tltl* which maps an $\mathrm{LTL}^c(\Gamma)$ formula θ to a $\mathrm{TLTL}^c(\Sigma, Op)$ formula obtained by replacing each atomic formula (a, g) by $a \wedge \theta_g$, (ϵ, g) by $\neg \bigvee_{c \in \Sigma} c \wedge \theta_g \wedge \neg(gSg \wedge gUg)$ and g by $\neg \bigvee_{c \in \Sigma} c \wedge \theta_g \wedge (gSg \wedge gUg)$. We then have $L(\varphi) = \mathit{tw}(F(\tilde{\varphi})) = \mathit{tw}(F(\theta)) = L(\mathit{ltl\text{-}tltl}(\theta))$. So $\mathit{ltl\text{-}tltl}(\theta)$ is the $\mathrm{TLTL}^c(\Sigma, Op)$ formula equivalent to φ.

7 Recursive Continuous Input Determined Automata

We now consider "recursive" *CIDA*'s. The main motivation is to increase the expressive power of our automata, as well as to characterize the expressiveness of recursive temporal logics which occur naturally in the real-time settings.

We define a *recursive* input-determined operator Δ over an alphabet Σ as a partial function from $(2^{\mathbb{R}_{\geq 0}} \times T\Sigma^* \times \mathbb{R}_{\geq 0})$ to $2^{\mathbb{R}_{\geq 0}}$, which is defined for tuples (X, σ, t) where $X \subseteq \mathbb{R}_{\geq 0}$, $\sigma \in T\Sigma^*$ and $t \in [0, \mathit{length}(\sigma)]$. Given a recursive operator Δ and a set $X \subseteq \mathbb{R}_{\geq 0}$, We denote by Δ_X, the operator whose semantics is given by $\Delta_X(\sigma, t) = \Delta(X, \sigma, t)$. We call a set X finitely varying if there exists a finitely varying function $f : [0, r] \rightarrow \{0, 1\}$ such that $X \subseteq [0, r]$ and $f(t) = 1$ if and only if $t \in X$. We call a recursive operator Δ *finitely varying* if for every finitely varying set X, Δ_X is a finitely varying operator.

Given a timed word σ in $T\Sigma^*$ and a $t \in [0, \mathit{length}(\sigma)]$ we call the pair (σ, t) a *floating timed word* over Σ. A floating timed language is then a set of floating timed words. We will use the notation Σ' for $(\Sigma \cup \{\epsilon\}) \times \{0, 1\}$. Given $\sigma' \in T\Sigma'^*$, we denote by σ the timed word obtained from σ' by projecting away the $\{0, 1\}$ component from each pair and then dropping any ϵ's in the resulting word. A timed word σ' over the alphabet Σ' which contains exactly one symbol from $(\Sigma \cup \{\epsilon\}) \times \{1\}$, and whose last symbol is from $\Sigma \times \{0, 1\}$, defines the floating timed word (σ, t) where t is the time of the unique action which has a 1-extension. We use *fw* to denote the (partial) map which given a timed word σ' over Σ' returns (σ, t) and extend it to apply to timed languages over Σ' in the natural way.

Let Σ be an alphabet and Op be a set of input determined operators. Given $\Delta \in Op$, we use the notation Δ' for the operator over Σ' with the semantics $\Delta'(\sigma', t) = \Delta(\sigma, t)$. We use the notation Op' to denote the set $\{\Delta' \mid \Delta \in Op\}$. We now define a *floating CIDA* over (Σ, Op) to be a *CIDA* over (Σ', Op'). We define the floating language of a floating *CIDA* $\mathcal{B}$, denoted $L^{fl}(\mathcal{B})$, as $fw(L(\mathcal{B}))$.

We define the recursive continuous input determined automata (*rec-CIDA*'s) and the floating recursive continuous input determined automata (*frec-CIDA*'s) over an alphabet Σ and a set of recursive operators Rop based on Σ, as the union of level i *rec-CIDA*'s and level i *frec-CIDA*'s, for all $i \in \mathbb{N}$, respectively.

- A level 0 *rec-CIDA* $\mathcal{A}$ is a *CIDA* over Σ that uses only the guard $\top$. It accepts the timed language $L(\mathcal{A})$. A level 0 *frec-CIDA* $\mathcal{B}$ is a floating *CIDA* over Σ which uses only the guard $\top$. It accepts the floating language $L^{fl}(\mathcal{B})$.
- Let C be a finite collection of *frec-CIDA*'s of level i or less over (Σ, Rop). Let Op be the set of operators $\{\Delta_{\mathcal{B}} \mid \Delta \in Rop, \mathcal{B} \in C\}$, where the semantics of each $\Delta_{\mathcal{B}}$ is defined as follows. Let $pos(\sigma, \mathcal{B}) = \{t \in [0, \mathit{length}(\sigma)] \mid (\sigma, t) \in L^{fl}(\mathcal{B})\}$. Then $\Delta_{\mathcal{B}}(\sigma, t) = \Delta(pos(\sigma, \mathcal{B}), \sigma, t)$. We say that an operator $\Delta_{\mathcal{B}}$ is of level j if $\mathcal{B}$ is a level j *frec-CIDA*. A level $i + 1$ *rec-CIDA*(Σ, Rop) is a *CIDA*(Σ, Op) which uses at least one operator of level i. And a level $i + 1$ *frec-CIDA*(Σ, Rop) is a floating *CIDA*(Σ, Op) which uses at least one operator of level i.

We now introduce the recursive version of TMSO^c and show that it characterizes the class of timed languages defined by *rec-CIDA*. Given an alphabet Σ and a set of recursive operators Rop, the set of formulas of *rec-TMSO*$^c(\Sigma, Rop)$ are defined inductively as:

$$\varphi ::= Q_a(x) \mid \Delta^I_\psi(x) \mid x < y \mid x \in X \mid \neg\varphi \mid \varphi \vee \varphi \mid \exists x\varphi \mid \exists X\varphi,$$

where $a \in \Sigma$, $\Delta \in Rop$, $I \in \mathcal{I}_{\mathbb{Q}}$ and ψ is a *rec-TMSO*^{c} formula with a single free variable z.

The logic is interpreted over timed words in $T\Sigma^*$. If φ contains no predicates of the form "$\Delta_\psi^I(x)$", then $\sigma, \mathbb{I} \models \varphi$ is defined as for TMSO^c. Inductively we assume that $\sigma, \mathbb{I} \models \psi$ is defined where ψ has a single free variable z. Let $pos(\sigma, \psi) = \{t \mid \sigma, [t/z] \models \psi\}$ be the set of interpretations of z which make ψ true in σ. We then consider Δ_ψ as an operator with the semantics $\Delta_\psi(\sigma, t) = \Delta(pos(\sigma, \psi), \sigma, t)$. The rest of the interpretation is similar to TMSO^c.

We note that each *rec-TMSO*$^c(\Sigma, Rop)$ formula can be viewed as a TMSO$^c(\Sigma, Op)$ formula where Op is the set of Δ_ψ's which have a *top-level* occurrence, i.e., they are not in the scope of any other Δ operator.

A *rec-TMSO*$^c(\Sigma, Rop)$ sentence φ defines the language $L(\varphi) = \{\sigma \in T\Sigma^* \mid \sigma \models \varphi\}$. A *rec-TMSO*$^c(\Sigma, Rop)$ formula ψ with one free variable z defines the floating language $L^{fl}(\psi) = \{(\sigma, t) \mid \sigma, [t/z] \models \psi\}$. We have the following characterization.

Theorem 7. *Let Rop be a set of finitely varying recursive operators and Σ be a finite alphabet. $L \subseteq T\Sigma^*$ is accepted by a rec-CIDA over (Σ, Rop) iff L is definable by a rec-TMSO$^c(\Sigma, Rop)$ sentence.*

We now define a recursive timed temporal logic along the lines of [6] and show that it is expressively complete. It is similar to the logic TLTL^c and is parameterized by an alphabet Σ and a set of recursive input-determined operators Rop, and is denoted *rec-TLTL*$^c(\Sigma, Rop)$. The syntax of the logic is given by

$$\theta ::= a \mid \Delta_\theta^I \mid (\theta U \theta) \mid (\theta S \theta) \mid \neg\theta \mid (\theta \vee \theta),$$

where $a \in \Sigma$, $\Delta \in Rop$ and $I \in \mathcal{I}_\mathbb{Q}$. The logic is interpreted over timed words in a manner similar to TLTL^c, where the satisfaction of the predicate Δ_θ^I by σ at t is equivalent to $\Delta(pos(\sigma, \theta), \sigma, t) \cap I = \emptyset$, and $pos(\sigma, \theta) = \{t \in \mathbb{R}_{\geq 0} \mid \sigma, t \models \theta\}$. Let us denote by *rec-TFO*$^c(\Sigma, Rop)$ the first order fragment of the logic *rec-TMSO*$^c(\Sigma, Rop)$. Then we have the following expressiveness result:

Theorem 8. *rec-TLTL$^c(\Sigma, Rop)$ is expressively equivalent to rec-TFO$^c(\Sigma, Rop)$.*

8 Expressive Completeness of MTL

As an application of the results in this paper we show that the logic Metric Temporal Logic (MTL^c) in the continuous semantics introduced in [7] is expressively equivalent to *rec-TFO*^{c} for a suitably defined set of recursive input-determined operators. We define the logic MTL$^c(\Sigma)$ inductively as below:

$$\theta ::= a \mid (\theta U_I \theta) \mid (\theta S_I \theta) \mid \neg\theta \mid (\theta \vee \theta),$$

where $a \in \Sigma$ and $I \in \mathcal{I}_\mathbb{Q}$. The modalities U_I and S_I are interpreted as follows for a timed word σ and $t \in [0, length(\sigma)]$.

$$\sigma, t \models \theta U_I \eta \text{ iff } \exists t' \geq t : t' - t \in I, \sigma, t' \models \eta, \text{ and } \forall t'' : t < t'' < t', \sigma, t'' \models \theta.$$
$$\sigma, t \models \theta S_I \eta \text{ iff } \exists t' \leq t : t - t' \in I, \sigma, t' \models \eta, \text{ and } \forall t'' : t' < t'' < t, \sigma, t'' \models \theta.$$

We first observe that $\mathrm{MTL}^c(\Sigma)$ is expressively equivalent to its sublogic $\mathrm{MTL}^{c^{\Diamond}}(\Sigma)$ in which the modalities U_I and S_I are replaced by the modalities U, S, $\Diamond_I$ and $\diamondminus_I$, where $\theta U \eta = \theta U_{(0,\infty)}\eta$, $\theta S \eta = \theta S_{(0,\infty)}\eta$, $\Diamond_I \theta = \top U_I \theta$ and $\diamondminus_I \theta = \top S_I \theta$. To show the equivalence we need to consider only the cases when $I = [l,l]$ and $I = (l,r)$. If $I = [l,l]$, then $\theta U_I \eta = \neg\Diamond_{(0,l)}\neg\theta \wedge \Diamond_{[l,l]}\eta$, otherwise $I = (l,r)$ in which case $\theta U_I \eta = \neg\Diamond_{(0,l]}\neg\theta \wedge \Diamond_{[l,l]}(\theta U \eta) \wedge \Diamond_{(l,r)}\eta$. Next we consider the logic *rec*-$TLTL^c(\Sigma, \{\Diamond, \diamondminus\})$ where the semantics of $\Diamond$ and $\diamondminus$ is defined as $\Diamond(X,\sigma,t) = \{t' - t \,|\, t' \geq t, t \in X\}$ and $\diamondminus(X,\sigma,t) = \{t - t' \,|\, t' \leq t, t \in X\}$. The logic $\mathrm{MTL}^{c^{\Diamond}}(\Sigma)$ is clearly expressively equivalent to *rec*-$TLTL^c(\Sigma, \{\Diamond, \diamondminus\})$, since the predicates $\Diamond_I \theta$ and $\Diamond_\theta^I$ are equivalent. Further $\Diamond$ and $\diamondminus$ are finitely varying recursive operators. Hence,

Theorem 9. $\mathrm{MTL}^c(\Sigma)$ *is expressively equivalent to rec-*$TFO^c(\Sigma, \{\Diamond, \diamondminus\})$.

References

1. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. R. Alur, T. Feder and T.A. Henzinger. The Benefits of Relaxing Punctuality. *Journal of the ACM*, 43(1):116–146, 1996.
3. J. F. Raskin and P. Y. Schobbens. State Clock Logic: A Decidable Real-Time Logic. In HART, pages 33–47, 1997.
4. D. D'Souza and P. S. Thiagarajan. Product Interval Automata: A Subclass of Timed Automata. In FSTTCS, pages 60–71, 1999.
5. D. D'Souza and M. Raj Mohan. Eventual Timed Automata. In FSTTCS, pages 322-334, 2005.
6. D. D'Souza and N. Tabareau. On timed automata with input-determined guards. *FORMATS/FTRTFT*, volume 3253 of *LNCS*, 68-83, Springer, 2004.
7. R. Koymans. Specifying Real-Time Properties with Metric Temporal Logic. *Real-Time Systems*, 2(4):255-299, 1990.
8. J. Ouaknine and J. Worrel. On the Decidability of Metric Temporal Logic. In LICS, pages 188-197. IEEE Computer Society, 2005.
9. P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. In FSTTCS, pages 432-443, 2005.
10. P. Prabhakar and D. D'Souza. On the expressiveness of MTL with past operators. Technical Report IISc-CSA-TR-2006-5 Indian Institute of Science, Bangalore 560012, India, May, 2006. URL: `http://archive.csa.iisc.ernet.in/TR/2006/5/`
11. T. A. Henzinger and J. F. Raskin and P. Y. Schobbens. The Regular Real-Time Languages. In ICALP, volume 1443 of LNCS, pages 580-591. Springer, 1998.
12. A. M. Rabinovich. Finite variability interpretation of monadic logic of order. Theor. Comput. Sci., 275(1-2):111-125, 2002.
13. J. A. W. Kamp. Tense Logic and the Theory of Linear Order. PhD thesis, University of California, Los Angeles, California, 1968.
14. J. F. Raskin. Logics, Automata and Classical Theories for Deciding Real-Time. PhD thesis, FUNDP, Belgium, 1999.
15. F. Chevalier and D. D'Souza and P. Prabhakar. On continuous timed automata with input-determined guards. Technical Report IISc-CSA-TR-2006-7 Indian Institute of Science, Bangalore 560012, India, June, 2006. URL: `http://archive.csa.iisc.ernet.in/TR/2006/7/`

Safely Freezing LTL

Ranko Lazić*

Department of Computer Science, University of Warwick, UK

Abstract. We consider the safety fragment of linear temporal logic with the freeze quantifier. The freeze quantifier is used to store a value from an infinite domain in a register for later comparison with other such values. We show that, for one register, satisfiability, refinement and model checking problems are decidable. The main result in the paper is that satisfiability is ExpSpace-complete. The proof of ExpSpace-membership involves a translation to a new class of faulty counter automata. We also show that refinement and model checking are not primitive recursive, and that dropping the safety restriction, adding past-time temporal operators, or adding one more register, each cause undecidability of all three decision problems.

1 Introduction

Logics and automata over finite alphabets, and their algorithmic properties, are central to formal specification and verification, and have been extensively studied: for a recent survey, see e.g. [1]. However, models which are not simply words or trees over finite alphabets arise in a variety of practical contexts. Examples include: computations of systems with unboundedly many locations or resources (e.g. [2]), XML documents with infinite data domains (e.g. [3,4]), and computations with discrete or continuous timestamps (e.g. [5]).

In this paper, we focus on models which are *data words*, i.e. words over a finite alphabet Σ where, at each position, there is also a datum from an infinite domain. The only data operation available is the equality predicate. Hence, formally, a data word is a word over Σ together with an equivalence relation on its indices.

A number of logical and automata formalisms over data words have been studied in the literature, including: linear temporal logic (LTL) with the freeze quantifier [6,7,2,8], first-order logic (FO) [3], and automata with registers or pebbles [9,10]. Much introductory and motivational material (including the research context and applications), as well as results on relative expressiveness, can be found in those references. The freeze quantifier ($\downarrow$) enables a word index to be stored in a register, and later tested for equivalence with the current index. For example, $\mathtt{G}\downarrow_1 \mathtt{X}\,\mathtt{G}\,\neg(\uparrow_1\sim)$ (where 1 denotes the first register and $\sim$ the equivalence relation) expresses the 'nonces property': that any two data in the model are distinct. In FO, the same property is expressible as $\forall x\,\forall y\,(x < y \Rightarrow \neg\, x \sim y)$.

[*] Supported by grants from the EPSRC (GR/S52759/01) and the Intel Corporation. Part of this research was done while visiting LSV, ENS Cachan. Also affiliated to the Mathematical Institute, Belgrade.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 381–392, 2006.

The main known decidability results for logics over data words are for: satisfiability for LTL with the freeze quantifier and 1 register, over finite data words [8]; and satisfiability for FO with 2 variables and predicates $\sim$, $<$ and $+1$, over finite and over infinite data words [3].

We consider the *safety* fragment of LTL with the freeze quantifier, designed for expressing safety properties. Formally, ϕ is a safety formula iff it does not contain an occurence of the eventually ($\mathtt{F}$) or until ($\mathtt{U}$) operator under an even number of negations. For example, the nonces sentence above is in the safety fragment, and also $\mathtt{G}(\mathtt{free} \Rightarrow {\downarrow_1} \mathtt{X} \neg(\neg(\mathtt{alloc} \wedge {\uparrow_1}{\sim}) \mathtt{U} \mathtt{access} \wedge {\uparrow_1}{\sim}))$ which states that, after being freed, a location is not accessed before being allocated.

Over finite words, $\phi \mathtt{U} \psi$ is equivalent to $\neg(\neg\psi \mathtt{U} \neg\psi \wedge (\neg\phi \vee \mathtt{X} \bot))$ (since $\mathtt{X} \bot$ is true only at the last index), so the full logic is translatable (in polynomial space) to the safety fragment. Hence, in this paper, we focus on infinite data words. For any safety sentence ϕ, the set of all infinite data words which satisfy ϕ is a safety property, i.e. closed under limits of finite prefixes [11]. Since the safety fragment is not closed under negation, there are three central decision problems to consider: satisfiability, refinement (whether a given sentence implies another given sentence), and model checking (whether each data word accepted by a given register automaton satisfies a given sentence). A fragment for which all three problems are decidable is said to be *fully decidable* [12].

The main result in the paper is that satisfiability for the safety fragment with 1 register is ExpSpace-complete. Membership of ExpSpace should be compared with the following: the full fragment with 1 register is undecidable over infinite data words (by [8, Theorem 15]), the safety fragment with 1 register is not primitive recursive over finite data words (by the remarks above and [8, Theorem 15]),[1] the fragment of FO shown decidable in [3] is at least as hard as reachability for Petri nets (for which elementarity is a long-standing open problem), and primitive recursiveness is not known for the safety fragment of Metric temporal logic [12]. The only previously known logics over data words with elementary satisfiability are: FO with 2 variables and predicates $\sim$ and $<$, which is NExpTime-complete over finite data words [14]; FO with 2 variables and predicates $\sim$ and $+1$, which is in 2NExpTime over finite data words [3]; and the flat fragment of LTL with the freeze quantifier, which is PSpace-complete over infinite data words [7, Corollary 4]. The absences of the $+1$ and $<$ predicates, and flatness, are severe restrictions. For example, the nonces property is not expressible in the flat fragment.

The proof of ExpSpace-membership is by reducing, in polynomial space and via alternating register automata, to nonemptiness for a new class of counter automata with incrementing errors. The latter have $\langle \mathtt{transf}, f \rangle$ instructions, where f is a mapping from counters to sets of counters. Such an instruction nondeterministically distributes the value of each counter c over the set of counters $f(c)$. The $\langle \mathtt{transf}, f \rangle$ instructions enable the counter automata computed from the logical formulae to have transition relations with no ϵ-cycles and all locations

[1] Recall the Ritchie-Cobham Property [13, page 297]: a decision problem is primitive recursive iff it is solvable in primitive recursive time/space.

accepting. Nonemptiness then amounts to existence of an infinite sequence of transitions from the initial state. By adapting and substantially extending the recent proof that termination for channel machines with insertion errors and emptiness testing is primitive recursive [15], we show how to compute, for any counter automaton from the class, a positive integer such that if the automaton has a sequence of transitions of that length from the initial state, then it has an infinite sequence. (The machines in [15] do not have an equivalent of the $\langle \mathtt{transf}, f\rangle$ instructions.) By Savitch's Theorem, we obtain that the nonemptiness problem is in ExpSpace. Although the counter automata computed from the logical formulae are of exponential size, the precise form of the integer bounds enables us to deduce that satisfiability for the safety fragment with 1 register is also in ExpSpace.

We also show that, for the safety fragment with 1 register, refinement and model checking are decidable. They have the validity problem as a special case, which can be shown not primitive recursive. Finally, we observe that dropping the safety restriction, adding the past-time operator $\mathtt{F}^{-1}$, or adding a register, each cause undecidability of satisfiability, refinement and model checking.

2 Preliminaries

2.1 LTL over Data Words

$\mathrm{LTL}^{\downarrow}(\sim;\mathcal{O})$ will denote the linear temporal logic with the freeze quantifier, the predicate $\sim$, and temporal operators in the set $\mathcal{O} \subseteq \{\mathtt{X}, \mathtt{X}^{-1}, \mathtt{F}, \mathtt{F}^{-1}, \mathtt{U}, \mathtt{U}^{-1}\}$. Each formula is over a finite alphabet Σ. Atomic propositions a are elements of Σ, r ranges over $\mathbb{N} \setminus \{0\}$, and $\mathtt{O}$ ranges over $\mathcal{O}$.

$$\phi ::= \top \mid a \mid {\uparrow_r}{\sim} \mid \neg\phi \mid \phi \wedge \phi \mid \mathtt{O}(\phi, \ldots, \phi) \mid {\downarrow_r}\, \phi$$

Models of $\mathrm{LTL}^{\downarrow}(\sim;\mathcal{O})$ are infinite *data words*. An infinite data word σ over a finite alphabet Σ is an infinite word $\sigma(0)\,\sigma(1)\,\cdots$ over Σ together with an interpretation of $\sim$ as an equivalence relation $\sim^{\sigma}$ on $\mathbb{N}$. Let $\Sigma^{\omega}(\sim)$ denote the set of all infinite data words over Σ. For $\sigma \in \Sigma^{\omega}(\sim)$, let $\mathrm{str}(\sigma)$ be the underlying word $\sigma(0)\,\sigma(1)\,\cdots$.

A *register valuation* v is a finite partial map from $\mathbb{N} \setminus \{0\}$ to $\mathbb{N}$. An undefined register value in an atomic formula will make the latter false. Undefined register values will be used for initial automata states. The satisfaction relation $\models$ is defined as follows. The temporal operators other than $\mathtt{X}$ and $\mathtt{U}$ are interpreted as expected. We also omit the Boolean cases.

$$\begin{array}{ll} \sigma, i \models_v a \overset{\text{def}}{\Leftrightarrow} \sigma(i) = a & \sigma, i \models_v {\uparrow_r}{\sim} \overset{\text{def}}{\Leftrightarrow} r \in \mathrm{dom}(v) \text{ and } v(r) \sim^{\sigma} i \\ \sigma, i \models_v \mathtt{X}\phi \overset{\text{def}}{\Leftrightarrow} \sigma, i+1 \models_v \phi & \sigma, i \models_v {\downarrow_r}\phi \overset{\text{def}}{\Leftrightarrow} \sigma, i \models_{v[r\mapsto i]} \phi \end{array}$$

$$\sigma, i \models_v \phi\mathtt{U}\psi \overset{\text{def}}{\Leftrightarrow} \text{for some } j \geq i,\ \sigma, j \models_v \psi \text{ and for all } i \leq j' < j,\ \sigma, j' \models_v \phi$$

A sentence ϕ over Σ is said to be satisfied by $\sigma \in \Sigma^{\omega}(\sim)$ iff $\sigma, 0 \models_{\emptyset} \phi$.

As $\mathtt{F}\phi$ is equivalent to $\top\mathtt{U}\phi$, $\mathtt{F}$ can be omitted from any set of temporal operators which contains $\mathtt{U}$, and the same holds for the past-time versions $\mathtt{F}^{-1}$

and $\mathtt{U}^{-1}$. As usual, we regard $\mathtt{G}$ and $\mathtt{G}^{-1}$ as abbreviations for $\neg\mathtt{F}\neg$ and $\neg\mathtt{F}^{-1}\neg$. An occurence of $\uparrow_r\!\sim$ within the scope of a freeze quantifier $\downarrow_r$ is bound by it; otherwise, it is free. A sentence is a formula with no free occurence of any $\uparrow_r\!\sim$.

A formula is in the *safety* [resp. *co-safety*] fragment iff it does not contain an occurence of $\mathtt{F}$ or $\mathtt{U}$ under an even [resp. odd] number of negations $\neg$. Hence, ϕ is a safety formula iff $\neg\phi$ is a co-safety formula, and vice-versa. $\mathrm{LTL}^{\downarrow}_n(\sim;\mathcal{O})$ is the fragment of $\mathrm{LTL}^{\downarrow}(\sim;\mathcal{O})$ with n registers, i.e. where $r \in \{1,\ldots,n\}$.

2.2 Register Automata

We shall define (one-way) alternating register automata over infinite data words, and nondeterministic register automata as a subclass. We shall only need to consider Büchi acceptance.

Following a standard approach to formalising alternation [16], we work with transition formulae in which disjunctions and conjunctions express existential and universal branching (respectively). Suppose Q is a finite set of locations, and $n \in \mathbb{N}$ specifies the number of registers. The set $\Phi(Q,n)$ of all *transition formulae* with respect to Q and n is defined as follows ($r \in \{1,\ldots,n\}$ and $q \in Q$):

$$\varphi \;::=\; \top \mid \bot \mid \uparrow_r\!\sim \mid \uparrow_r\!\not\sim \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \downarrow_r \varphi \mid q$$

A transition formula is *locationless* iff it has no subformula of the form q. Otherwise, it is *locationful.* For any locationless transition formula φ, let $\bar{\varphi}$ denote its dual, obtained by replacing any atomic subformula by its negation, and interchanging $\wedge$ and $\vee$.

An alternating *register automaton (RA)* $\mathcal{A}$ is a tuple $\langle \Sigma, Q, q_I, n, \delta, F\rangle$ such that: Σ is a finite alphabet, Q is a finite set of locations, $q_I \in Q$ is the initial location, $n \in \mathbb{N}$ is the number of registers (given in unary), $\delta : Q \times \Sigma \to \Phi(Q,n)$ is the transition function, and $F \subseteq Q$ is the set of accepting locations.

A state of $\mathcal{A}$ is a triple $\langle i, q, v\rangle$ where $i \in \mathbb{N}$ specifies a word index, $q \in locs$, and v is a valuation for n registers. To interpret transition formulae in $\mathcal{A}$, suppose $\sigma \in \Sigma^{\omega}(\sim)$. We define a relation $S \models^{\sigma,i}_{v} \varphi$, which means that a finite set of states S satisfies a transition formula φ at index i of σ and with register valuation v. The definition is recursive over φ, where the Boolean cases are standard, and dual clauses are treated as expected:

$$S \models^{\sigma,i}_{v} \uparrow_r\!\sim \;\overset{\text{def}}{\Leftrightarrow}\; v(r) \sim^{\sigma} i \text{ and } v(r) \text{ is defined}$$
$$S \models^{\sigma,i}_{v} \downarrow_r\varphi \;\overset{\text{def}}{\Leftrightarrow}\; S \models^{\sigma,i}_{v[r\mapsto i]} \varphi \qquad S \models^{\sigma,i}_{v} q \;\overset{\text{def}}{\Leftrightarrow}\; \langle i+1, q, v\rangle \in S$$

A run of $\mathcal{A}$ over σ is a directed acyclic graph $\langle G, \to\rangle$ such that: G is a set of states of $\mathcal{A}$; $\langle 0, q_I, \emptyset\rangle$ (the initial state) is the unique vertex with no predecessors, and all other vertices are reachable from it; for any state $\langle i, q, v\rangle$, the set of all its successors is a minimal S satisfying $S \models^{\sigma,i}_{v} \delta(q, \sigma(i))$. For any $i \in \mathbb{N}$, let $G(i)$ be the set of all states $\langle i, q, v\rangle$ in G. Observe that $G(i+1)$ is the set of all successors of states in $G(i)$, so $G(i)$ is the i^{th} level of $\langle G, \to\rangle$.

A run $\langle G, \to\rangle$ is accepting iff, along each infinite path from the root, an accepting location occurs infinitely often. We say that $\mathcal{A}$ accepts σ iff $\mathcal{A}$ has an accepting run over σ. Note that any finite run is accepting. (Runs are finitely branching, so a run is finite iff it has no infinite paths from the root.) Note also that σ can be rejected because $\mathcal{A}$ has no runs over σ.

We say that $\mathcal{A}$ is nondeterministic (i.e. existential) iff any transition subformula which is a conjunction of locationful formulae is of the form $(\varphi \vee \varphi') \wedge (\bar{\varphi} \vee \varphi'')$ where φ is locationless. The runs of such RA are sequences of states.

2.3 Counter Automata

We define Automata over infinite words, with Counters which are elements of a Powerset, ε transitions, Nondeterministic Transfers, Incrementing errors, and Büchi acceptance: for short, *IPCANT*. Let $\mathcal{P}^+(X)$ denote the set of all nonempty subsets of X. An IPCANT $\mathcal{C}$ is a tuple $\langle \Sigma, Q, q_I, X, C, \delta, F\rangle$ such that: Σ is a finite alphabet, Q is a finite set of locations, q_I is the initial location, X is a finite set (called the *basis*), $C \subseteq \mathcal{P}^+(X)$ is a nonempty set of counters, $\delta \subseteq Q \times (\Sigma \uplus \{\varepsilon\}) \times L \times Q$ is the transition relation (given as a list), and $F \subseteq Q$ is the set of accepting locations. The instruction set L consists of: $\langle \texttt{inc}, c\rangle$ and $\langle \texttt{dec}, c\rangle$ for each $c \in C$, and $\langle \texttt{transf}, f\rangle$ for each mapping $f : C \to \mathcal{P}(C)$ which is *distributive* as follows:

if $c \in C$, $c \subseteq \bigcup_{i=1}^{k} c_i$, and $c'_i \in f(c_i)$ for each $i = 1, \ldots, k$,
then there exists $c' \in f(c)$ with $c' \subseteq \bigcup_{i=1}^{k} c'_i$.

A state of $\mathcal{C}$ is a pair $\langle q, v\rangle$ such that $q \in Q$ and v is a counter valuation, where counter valuations are mappings $C \to \mathbb{N}$. A state $\langle q, v\rangle$ has an error-free transition $\xrightarrow{w,l}_{\surd}$ to $\langle q', v'\rangle$ iff $\langle q, w, l, q'\rangle \in \delta$ and v' can be obtained from v using l. The instructions $\langle \texttt{inc}, c\rangle$ and $\langle \texttt{dec}, c\rangle$ have the standard interpretations, where $\langle \texttt{dec}, c\rangle$ is firable iff $v(c) > 0$. An instruction $\langle \texttt{transf}, f\rangle$ transfers the value of each counter c to the counters in $f(c)$, nondeterministically distributing it. More precisely, v' can be obtained from v using $\langle \texttt{transf}, f\rangle$ iff there exist $d^c_{c'} \geq 0$ for each $c \in C$ and $c' \in f(c)$, such that $v(c) = \sum_{c' \in f(c)} d^c_{c'}$ for each $c \in C$, and $v'(c') = \sum_{f(c) \ni c'} d^c_{c'}$ for each $c' \in C$.

For counter valuations v and $v_{\surd}$, we write $v \leq v_{\surd}$ iff, for all c, $v(c) \leq v_{\surd}(c)$. To allow transitions of IPCANT to contain arbitrary errors which increment one or more counters, we define $\langle q, v\rangle \xrightarrow{w,l} \langle q', v'\rangle$ iff there exist $v_{\surd}$ and $v'_{\surd}$ such that $v \leq v_{\surd}$, $\langle q, v_{\surd}\rangle \xrightarrow{w,l}_{\surd} \langle q', v'_{\surd}\rangle$, and $v'_{\surd} \leq v'$.

A run of $\mathcal{C}$ over $w \in \Sigma^\omega$ is an infinite sequence $\langle q_0, v_0\rangle \xrightarrow{w_0,l_0} \langle q_1, v_1\rangle \xrightarrow{w_1,l_1} \cdots$ where $\langle q_0, v_0\rangle$ is the initial state $\langle q_I, \underline{0}\rangle$, and $w = w_0 w_1 \ldots$. Such a run is accepting iff $q_i \in F$ for infinitely many i. $\mathcal{C}$ accepts w iff it has an accepting run over w.

Example 1. Given $Y \subseteq X$, let $f_Y(c) = \emptyset$ if $c \cap Y \neq \emptyset$, and $f_Y(c) = \{c\}$ otherwise. Observe that f_Y is distributive. The instruction $\langle \texttt{transf}, f_Y\rangle$ is firable iff each counter c which intersects Y is zero, and it does not change the value of any counter. Hence, we may write $\langle \texttt{iszero}, Y\rangle$ instead of $\langle \texttt{transf}, f_Y\rangle$.

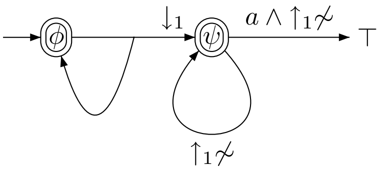
3 From LTL$^{\downarrow}$ to Register Automata

A sentence ϕ of LTL$^{\downarrow}(\sim;\mathcal{O})$ is *equivalent* to an RA $\mathcal{A}$ iff they are over the same alphabet Σ and, for each $\sigma \in \Sigma^{\omega}(\sim)$, ϕ is satisfied by σ iff $\mathcal{A}$ accepts σ. It was shown already in [9, Proposition 5] that no nondeterministic RA is equivalent to the nonces sentence $\mathtt{G}\downarrow_1\mathtt{X}\,\mathtt{G}\,\neg(\uparrow_1\sim)$.

Theorem 1. *Given a sentence ϕ of LTL$^{\downarrow}(\sim;\mathtt{X},\mathtt{U})$ with alphabet Σ, an equivalent alternating RA $\mathcal{A}^{\Sigma}_{\phi}$ with equally many registers and whose number of locations is linear in $|\phi|$ is computable in logarithmic space. If ϕ is a safety [resp. co-safety] formula, then $\mathcal{A}^{\Sigma}_{\phi}$ can be constructed with all [resp. no] locations accepting.*

Proof. By adapting the proof of [8, Theorem 8], which shows how to compute in logarithmic space an equivalent alternating RA with weak parity acceptance. □

Example 2. Consider the safety LTL$^{\downarrow}_{1}(\sim;\mathtt{X},\mathtt{U})$ sentence $\phi = \mathtt{G}\downarrow_1\mathtt{X}\,\psi$, where $\psi = \neg(\neg a\,\mathtt{U}\uparrow_1\sim)$, over $\Sigma = \{a, b\}$. ϕ states that a must occur between each pair of $\sim$-equivalent positions. The following is an equivalent alternating RA.



Transitions unlabelled by a letter can be taken with any letter. The fork represents a conjunctive branching, which spawns a new thread. When a state performs the transition leading to $\top$, it terminates, i.e. has no successors at the next level.

A set of infinite words is considered to be a safety property [11] iff each infinite word not belonging to the set has a 'doomed' finite prefix, i.e. such that all infinite words which extend it also do not belong to the set. It is a corollary of Theorem 1 that safety sentences of LTL$^{\downarrow}(\sim;\mathtt{X},\mathtt{U})$ induce safety properties.

4 Satisfiability

In this section, we show that safety LTL$^{\downarrow}_{1}(\sim;\mathtt{X},\mathtt{U})$ satisfiability is ExpSpace-complete. Dropping the safety restriction, adding $\mathtt{F}^{-1}$, or adding one more register, each produce undecidability. More precisely, LTL$^{\downarrow}_{1}(\sim;\mathtt{X},\mathtt{U})$ satisfiability is Π^0_1-complete by [8, Corollary 13 and Theorem 15]. We have Π^0_1-hardness for safety LTL$^{\downarrow}_{1}(\sim;\mathtt{X},\mathtt{U},\mathtt{F}^{-1})$ and safety LTL$^{\downarrow}_{2}(\sim;\mathtt{X},\mathtt{U})$ by the proof of [8, Theorem 17], and Π^0_1-membership by Theorem 1 (which can be extended to past-time operators and two-way RA) and König's Lemma.

4.1 Membership

By Theorem 1 and Theorem 2 below, for safety LTL$^{\downarrow}_{1}(\sim;\mathtt{X},\mathtt{U})$ sentences ϕ over alphabets Σ, an IPCANT $\mathcal{C}_{\mathcal{A}^{\Sigma}_{\phi}}$ is computable in polynomial space, whose basis size is linear in $|\phi|$, which satisfies the assumptions in Theorem 3 below, and which is nonempty iff ϕ is satisfiable. Hence, by Theorem 3, safety LTL$^{\downarrow}_{1}(\sim;\mathtt{X},\mathtt{U})$ satisfiability is in ExpSpace.

Let us say that an IPCANT $\mathcal{C}$ *corresponds* to an RA $\mathcal{A}$ iff they have the same alphabet Σ and, for each $w \in \Sigma^\omega$, $\mathcal{C}$ accepts w iff $\mathcal{A}$ accepts some σ with $\mathrm{str}(\sigma) = w$. In particular, $\mathcal{C}$ is nonempty iff $\mathcal{A}$ is.

Theorem 2. *Given an alternating RA $\mathcal{A}$ which has* 1 *register and all locations accepting, a corresponding IPCANT $\mathcal{C}_\mathcal{A}$ whose basis size is linear in the number of locations of $\mathcal{A}$, whose transition relation contains no cycles of ε transitions, and which has all locations accepting, is computable in polynomial space.*

Proof. By the proof of [8, Theorem 12], a corresponding IPCANT $\mathcal{C}'_\mathcal{A}$ with nondeterministic transfer instructions only of the form $\langle \mathtt{iszero}, Y\rangle$ (see Example 1), and which has all locations accepting, is computable in polynomial space. Writing Q for the set of locations of $\mathcal{A}$, the basis of $\mathcal{C}'_\mathcal{A}$ is $Q \uplus \widehat{Q}$, where $\widehat{Q} = \{\widehat{q} : q \in Q\}$. The counters of $\mathcal{C}'_\mathcal{A}$ are $Q^\dagger$ and $\widehat{Q^\dagger}$ as $Q^\dagger$ ranges over $\mathcal{P}^+(Q)$. After $\mathcal{C}'_\mathcal{A}$ simulates a run of $\mathcal{A}$ over a finite data word τ, the counters $Q^\dagger$ represent the last level of the run. Lifting $\sim^\tau$ to the level and projecting onto Q gives a multiset over $\mathcal{P}^+(Q)$, and each counter $Q^\dagger$ stores the number of occurences of $Q^\dagger$ in the multiset. The instruction $\langle \mathtt{iszero}, Q\rangle$ [resp. $\langle \mathtt{iszero}, \widehat{Q}\rangle$] is used in $\mathcal{C}'_\mathcal{A}$ to check whether all the counters $Q^\dagger$ [resp. $\widehat{Q^\dagger}$] are zero. $\mathcal{C}'_\mathcal{A}$ accepts w by an error-free run iff $\mathcal{A}$ accepts some σ with $\mathrm{str}(\sigma) = w$. Finally, whenever $\mathcal{C}'_\mathcal{A}$ accepts w by a run possibly with incrementing errors, it is also accepts w by an error-free run.

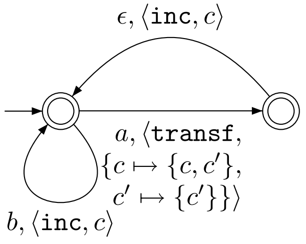
In general, the transition relation of $\mathcal{C}'_\mathcal{A}$ contains cycles of ε transitions of the following form. Given $Q^\dagger \in \mathcal{P}^+(Q)$ and $a \in \Sigma$, the transition function of $\mathcal{A}$ determines a subset $\mathcal{Q}(Q^\dagger, a)$ of $(\mathcal{P}(Q))^2$. The ith pass through the cycle for $Q^\dagger$ and a consists of decrementing the counter $Q^\dagger$, nondeterministically choosing $\langle Q_i^=, Q_i^{\neq}\rangle \in \mathcal{Q}(Q^\dagger, a)$, and incrementing the counter $\widehat{Q_i^{\neq}}$ if $Q_i^{\neq} \neq \emptyset$. The union of all $Q_i^=$ is stored in the control of $\mathcal{C}'_\mathcal{A}$.

To compute $\mathcal{C}_\mathcal{A}$ as required in polynomial space, we modify the computation of $\mathcal{C}'_\mathcal{A}$ to eliminate the cycles of ε transitions as follows. The basis of $\mathcal{C}_\mathcal{A}$ is $\{*\} \uplus Q \uplus \{\widetilde{\overline{*}}\} \uplus \overline{Q} \uplus \widetilde{Q}$. The counters of $\mathcal{C}_\mathcal{A}$ are $\{*\} \uplus Q^\dagger$ and $\{\widetilde{\overline{*}}\} \uplus \overline{Q^=} \uplus \widetilde{Q^{\neq}}$ as $Q^\dagger$, $Q^=$ and $Q^{\neq}$ range over $\mathcal{P}(Q)$. The counters $\{*\} \uplus Q^\dagger$ for $Q^\dagger \neq \emptyset$ correspond to the counters $Q^\dagger$ of $\mathcal{C}'_\mathcal{A}$. The remaining counters of $\mathcal{C}_\mathcal{A}$ are auxiliary. Now, suppose $a \in \Sigma$. Instead of the cycles of ε transitions, $\mathcal{C}_\mathcal{A}$ performs $\langle \mathtt{transf}, f\rangle$, where

$$f(\{*\} \uplus Q^\dagger) = \{\{\widetilde{\overline{*}}\} \uplus \overline{Q^=} \uplus \widetilde{Q^{\neq}} : \langle Q^=, Q^{\neq}\rangle \in \mathcal{Q}(Q^\dagger, a)\}$$
$$f(\{\widetilde{\overline{*}}\} \uplus \overline{Q^=} \uplus \widetilde{Q^{\neq}}) = \{\{\widetilde{\overline{*}}\} \uplus \overline{Q^=} \uplus \widetilde{Q^{\neq}}\}$$

(Note that $\mathcal{Q}(\emptyset, a) = \{\langle \emptyset, \emptyset\rangle\}$.) The next stage is that $\mathcal{C}_\mathcal{A}$ computes the union of all $Q^=$ such that some counter $\{\widetilde{\overline{*}}\} \uplus \overline{Q^=} \uplus \widetilde{Q^{\neq}}$ is nonzero, as follows. It chooses a subset $Q^=_\cup$ of Q, performs $\langle \mathtt{iszero}, \overline{Q \setminus Q^=_\cup}\rangle$, and checks whether, for each $q \in Q^=_\cup$, some counter $\{\widetilde{\overline{*}}\} \uplus \overline{Q^=} \uplus \widetilde{Q^{\neq}}$ with $Q^= \ni q$ is nonzero. $\mathcal{C}_\mathcal{A}$ then performs $\langle \mathtt{transf}, f'\rangle$, where $f'(\{\widetilde{\overline{*}}\} \uplus \overline{Q^=} \uplus \widetilde{Q^{\neq}}) = \{\{*\} \uplus Q^{\neq}\}$ and $f'(\{*\} \uplus Q^\dagger) = \{\{*\} \uplus Q^\dagger\}$. It remains to observe that, like $\mathcal{C}'_\mathcal{A}$, if $\mathcal{C}_\mathcal{A}$ accepts w by a run possibly with incrementing errors, it is also accepts w by an error-free run. □

Example 3. Simplifying $\mathcal{C}_{\mathcal{A}}$ for $\mathcal{A}$ in Example 2, we get the following IPCANT.


$\epsilon, \langle \mathtt{inc}, c \rangle$
$a, \langle \mathtt{transf},$
$\{c \mapsto \{c, c'\},$
$c' \mapsto \{c'\}\}\rangle$
$b, \langle \mathtt{inc}, c \rangle$

The counters are $c = \{*, \psi\}$ and $c' = \{*\}$. The counter c stores the number of $\sim$-classes of positions associated with the location ψ in a run level. Performing the transfer instruction represents a subset of threads choosing to terminate upon seeing a whose position is in a new class. Each $w \in \{a, b\}^\omega$ is accepted, because there exists σ with $\mathrm{str}(\sigma) = w$ and which is accepted by $\mathcal{A}$ (i.e. satisfies ϕ).

Theorem 3. *Nonemptiness of IPCANT whose transition relations contain no cycles of ε transitions and which have all locations accepting is decidable in space exponential in basis size and logarithmic in alphabet size and number of locations.*

Proof. Suppose $\mathcal{C} = \langle \Sigma, Q, q_I, X, C, \delta, F \rangle$ is as in the statement. We have that $\mathcal{C}$ is nonempty iff it has an infinite sequence of transitions from the initial state. Observe also that if $\mathcal{C}$ has an infinite sequence of transitions from the initial state, then it has such a sequence which is *lazy*, i.e. where incrementing errors occur only as $\langle \mathtt{dec}, c \rangle$ instructions which do not change the value of c.

We define positive integers α_i and U_i for $i = 0, \ldots, |X|$ as follows:

$$\alpha_0 = |Q| \qquad U_0 = 1 \qquad \alpha_{i+1} = 2(|X| - i)\alpha_i U_i^{|C|} \qquad U_{i+1} = 3\alpha_i U_i^{|C|}$$

Let $m = 2\alpha_{|X|} U_{|X|}^{|C|}$. We shall show that, if $\mathcal{C}$ has a lazy sequence of transitions of length $m-1$ from the initial state, then it has an infinite sequence. In such a lazy sequence S, each counter is at most $m-1$. For guessing S, it suffices to store only one transition at a time. Since $m < 2^{2^{2|X|^2+|X|} \log(3|Q|)}$, it follows by Savitch's Theorem that nonemptiness of $\mathcal{C}$ is decidable in space $2^{O(|X|^2)} O(\log(|\Sigma||Q|))$.

Suppose $\mathcal{C}$ has a lazy sequence of transitions $S = \langle q_1, v_1 \rangle \xrightarrow{w_1, l_1} \cdots \xrightarrow{w_{m-1}, l_{m-1}} \langle q_m, v_m \rangle$ from the initial state, but no infinite sequence. Let $q \in Q$ and $J_0 \subseteq \{1, \ldots, m\}$ be such that $|J_0| = m/\alpha_0 U_0^{|C|}$ and $q_j = q$ for each $j \in J_0$. We claim:

> There exists an enumeration $x_1, \ldots, x_{|X|}$ of X, and for $i = 1, \ldots, |X|$, mappings $u_i : C_i \to \{0, \ldots, U_i - 1\}$ where $C_i = \{c \in C : x_i \in c \wedge x_1, \ldots, x_{i-1} \notin c\}$, and subsets J_i of $\{1, \ldots, m\}$ of size $m/\alpha_i U_i^{|C|}$, such that for each $0 \leq i \leq |X|$ and $j \in J_i$, we have $q_j = q$ and $v_j(c) = u_{i'}(c)$ for all $c \in C_{i'}$ and $1 \leq i' \leq i$.

The claim holds for $i = 0$. Assume that $0 \leq i < |X|$ and that $x_{i'}$, $u_{i'}$ and $J_{i'}$ for $1 \leq i' \leq i$ have been picked so that the claim holds for i. Let us call a subsequence of S an *i-subsequence* iff there exist consecutive $j, j' \in J_i$ (i.e. where there is no $j'' \in J_i$ with $j < j'' < j'$) such that the subsequence begins at $\langle q_j, v_j \rangle$ and ends at $\langle q_{j'}, v_{j'} \rangle$. Consider the $m/2\alpha_i U_i^{|C|}$ shortest i-subsequences, and let $J'_i \subseteq J_i$ consist of their beginning positions. The length of the longest of those i-subsequences must be at most $2\alpha_i U_i^{|C|}$. Let $S^\dagger = \langle q_j, v_j \rangle \xrightarrow{w_j, l_j} \cdots \xrightarrow{w_{j'-1}, l_{j'-1}} \langle q_{j'}, v_{j'} \rangle$ be an i-subsequence with $j \in J'_i$. We have $q_j = q_{j'} = q$, $v_j(c) = v_{j'}(c) =$

$u_{i'}(c)$ for all $c \in C_{i'}$ and $1 \le i' \le i$, and $j' - j \le 2\alpha_i U_i^{|C|}$. In particular, $\sum_{i'=1}^{i} \sum_{c \in C_{i'}} v_{j'}(c) \le \sum_{i'=1}^{i} |C_{i'}| U_{i'}$.

Assume that, for each $x' \neq x_1, \ldots, x_i$, there exists $c_{x'}$ such that $x' \in c_{x'}$, $x_1, \ldots, x_i \notin c_{x'}$, and $v_j(c_{x'}) > 2\alpha_i U_i^{|C|} + \sum_{i'=1}^{i} |C_{i'}| U_{i'}$. Let H be a directed acyclic graph on $\{j, \ldots, j'\} \times C$, defined by letting the successors of $\langle j^\dagger, c^\dagger \rangle$ be: $\emptyset$, if $j^\dagger = j'$; $\{\langle j^\dagger + 1, c^\ddagger \rangle : c^\ddagger \in f(c^\dagger)\}$, if $l_{j^\dagger}$ is of the form $\langle \mathtt{transf}, f \rangle$; $\{\langle j^\dagger + 1, c^\dagger \rangle\}$, otherwise. Now, for $c \in C$ and $j^\dagger \in \{j, \ldots, j'\}$, let $H(c, j^\dagger)$ be the set of all $c^\dagger$ such that $\langle j^\dagger, c^\dagger \rangle$ is reachable in H from $\langle j, c \rangle$. We have $\sum_{c^\dagger \in H(c, j^\dagger)} v_{j^\dagger}(c^\dagger) \ge v_j(c) - (j^\dagger - j)$ by induction on $j^\dagger$. In particular, for each $x' \neq x_1, \ldots, x_i$, we have $\sum_{c^\dagger \in H(c_{x'}, j')} v_{j'}(c_{x'}) \ge v_j(c_{x'}) - (j' - j) > \sum_{i'=1}^{i} |C_{i'}| U_{i'}$, so H contains a path $H_{c_{x'}}$ from $\langle j, c_{x'} \rangle$ to some $\langle j', c^\dagger \rangle$ with $x_1, \ldots, x_i \notin c^\dagger$.

Consider any c with $x_1, \ldots, x_i \notin c$. We have $c \subseteq \bigcup \{c_{x'} : x' \in c\}$. By distributivity of nondeterministic transfer mappings and the definition of H, there exists a path H_c from $\langle j, c \rangle$ to some $\langle j', c^\dagger \rangle$ with $x_1, \ldots, x_i \notin c^\dagger$. For $j^\dagger \in \{j, \ldots, j'\}$, let $H_c(j^\dagger)$ be the counter at position $j^\dagger$ in H_c.

Given any v'_j such that $v'_j(c) = v_j(c)$ for all $c \in C_{i'}$ and $1 \le i' \le i$, by induction we can find $\langle q_j, v'_j \rangle \xrightarrow{w_j, l_j} \cdots \xrightarrow{w_{j'-1}, l_{j'-1}} \langle q_{j'}, v'_{j'} \rangle$ such that $v'_{j^\dagger}(c^\dagger) = v_{j^\dagger}(c^\dagger)$ for all $j^\dagger \in \{j+1, \ldots, j'\}$ and $c^\dagger \notin \{H_c(j^\dagger) : x_1, \ldots, x_i \notin c\}$. Since $v_j(c) = v_{j'}(c)$ for all $c \in C_{i'}$ and $1 \le i' \le i$, it follows that $\mathcal{C}$ has an infinite sequence of transitions, obtained by following S to position j, and then repeatedly simulating $S^\dagger$. That is a contradiction, so there exists $x' \neq x_1, \ldots, x_i$ such that $v_j(c) \le 2\alpha_i U_i^{|C|} + \sum_{i'=1}^{i} |C_{i'}| U_{i'} < U_{i+1}$ for all c with $x' \in c$ and $x_1, \ldots, x_i \notin c$.

Let $x_{i+1} \neq x_1, \ldots, x_i$ be such that there exists $J''_i \subseteq J'_i$ with $|J''_i| = m/\alpha_{i+1}$ and $v_j(c) < U_{i+1}$ for all $j \in J''_i$ and $c \in C_{i+1}$. Then let $u_{i+1} : C_{i+1} \to \{0, \ldots, U_{i+1} - 1\}$ be such that there exists $J_{i+1} \subseteq J''_i$ with $|J_{i+1}| = m/\alpha_{i+1} U_{i+1}^{|C|}$ and $v_j(c) = u_{i+1}(c)$ for all $j \in J_{i+1}$ and $c \in C_{i+1}$. That completes the inductive proof of the claim.

Since $m = 2\alpha_{|X|} U_{|X|}^{|C|}$, we have from the claim above that S contains two equal states, so $\mathcal{C}$ has an infinite sequence of transitions from the initial state. That is a contradiction, completing the proof. □

4.2 Hardness

Theorem 4. *Satisfiability for safety LTL*$_1^{\downarrow}(\sim; \mathtt{X}, \mathtt{U})$ *is* EXPSPACE*-hard.*

Proof. We shall show EXPSPACE-hardness by reducing from the Halting problem for Turing machines with exponentially long tapes. More precisely, a Turing machine $\mathcal{M}$ is a tuple $\langle \Sigma, a_B, Q, q_I, \delta \rangle$ such that: Σ is a finite alphabet, $a_B \in \Sigma$ denotes the blank symbol, Q is a finite set of locations, $q_I \in Q$ is the initial location, and $\delta : Q \times \Sigma \to Q \times (\Sigma \uplus \{\triangleleft, \triangleright\})$ is the transition function. If the size of $\mathcal{M}$ is n, we consider its computation on a tape of length 2^n. More formally, a state of $\mathcal{M}$ is of the form $\langle q, i, w \rangle$ where $q \in Q$ is the machine location, $0 \le i < 2^n$ is the head position, and $w \in \Sigma^{2^n}$ is the tape contents. The initial state is $\langle q_I, 0, a_B^{2^n} \rangle$. A state $\langle q, i, w \rangle$ has a transition iff neither $i = 0$ and $\delta(q, w(i))_2 = \triangleleft$, nor $i = 2^n - 1$

and $\delta(q, w(i))_2 = \triangleright$. In that case, we write $\langle q, i, w\rangle \to \langle \delta(q, w(i))_1, i', w'\rangle$ where the new head position i' and tape contents w' are determined as expected.

The following problem is ExpSpace-complete: given $\mathcal{M} = \langle \Sigma, a_B, Q, q_I, \delta\rangle$ of size n, to decide whether the computation from the initial state with tape length 2^n is infinite. We shall show that a sentence $\phi_{\mathcal{M}}$ of safety $\mathrm{LTL}_1^{\downarrow}(\sim; \mathtt{X}, \mathtt{U})$ can be computed in space logarithmic in n, such that the answer to the decision problem is 'yes' iff ϕ is satisfiable.

$\phi_{\mathcal{M}}$ will have alphabet $\tilde{\Sigma} = Q \uplus \{0_d, 1_d : d \in \{1, \ldots, n\}\} \uplus \{a, \widehat{a} : a \in \Sigma\}$. A state $\langle q, i, w\rangle$ is encoded by the word

$$q\, 0_1 \cdots 0_{n-1}\, 0_n\, w(0,i)\, 0_1 \cdots 0_{n-1}\, 1_n\, w(1,i) \cdots 1_1 \cdots 1_{n-1}\, 1_n\, w(2^n - 1, i)$$

where $w(i,i) = \widehat{w(i)}$, and $w(j,i) = w(j)$ for $j \neq i$.

The computation of $\mathcal{M}$ from the initial state with tape length 2^n is infinite iff there exists $\sigma \in \tilde{\Sigma}^{\omega}(\sim)$ such that:

(i) $\mathrm{str}(\sigma)$ is a sequence of encodings of states of $\mathcal{M}$;
(ii) $\mathrm{str}(\sigma)$ begins with the encoding of the initial state $\langle q_I, 0, a_B^{2^n}\rangle$;
(iii) for any two consecutive encodings in $\mathrm{str}(\sigma)$ of states $\langle q, i, w\rangle$ and $\langle q', i', w'\rangle$, we have $\langle q, i, w\rangle \to \langle q', i', w'\rangle$.

Hence, it suffices to construct $\phi_{\mathcal{M}}$ such that σ satisfies $\phi_{\mathcal{M}}$ iff (i)–(iii) hold and:

(iv) for any encoding in σ of a tape position, all the letters b_d and $w(j,i)$ are in the same class of $\sim^{\sigma}$;
(v) for any two encodings in σ of tape positions j and j' (occuring in one or two state encodings), their classes of $\sim^{\sigma}$ are the same iff $j = j'$.

Properties (iv) and (v) will be used to help navigation through σ in $\phi_{\mathcal{M}}$.

The most involved part of (i) is that the binary representation of any $j < 2^n - 1$ is followed by that of $j+1$. It is expressed by the following $\mathrm{LTL}_1^{\downarrow}(\sim; \mathtt{X}, \mathtt{U})$ sentence, which is in the safety fragment because $\Rightarrow$ implicitly negates its left-hand side. Let $\delta_d \stackrel{\mathrm{def}}{=} 0_d \vee 1_d$.

$$\begin{gathered}
\mathtt{G}\Bigg(\delta_1 \Rightarrow \textstyle\bigwedge_{d=1}^{n} \Bigg(\Big(\big(\bigvee_{d'=1}^{d-1} \delta_{d'}\big)\, \mathtt{U}\, \big(0_d \wedge \mathtt{X}\, ((\bigvee_{d'=d+1}^{n} 1_{d'})\, \mathtt{U} \bigvee_{a \in \Sigma} a \vee \widehat{a})\big)\Big) \Rightarrow \\
\Big(\textstyle\bigwedge_{d'=1}^{d-1} \bigwedge_{b=0}^{1} \neg\big(\neg\delta_{d'}\, \mathtt{U}\, (b_{d'} \wedge \mathtt{X}\, (\neg\delta_{d'}\, \mathtt{U}\, (1-b)_{d'}))\big)\Big) \wedge \\
\mathtt{X}\, \neg\Big(\neg\delta_1\, \mathtt{U}\, \big(\delta_1 \wedge ((\neg\delta_d\, \mathtt{U}\, 0_d) \vee \textstyle\bigvee_{d'=d+1}^{n} (\neg\delta_{d'}\, \mathtt{U}\, 1_{d'}))\big)\Big)\Bigg)\Bigg)
\end{gathered}$$

Expressing (ii) and (iv) is straightforward. For (iii), we use the presence of (iv) and (v) to avoid making $|\phi_{\mathcal{M}}|$ exponential in n. For (v), we specify that: in the encoding in σ of the first state, the classes of $\sim^{\sigma}$ for any two tape positions are distinct; for each encoding in σ of a tape position, its class of $\sim^{\sigma}$ occurs in the next state encoding; for any two encodings in σ of tape positions j and j' occuring in two consecutive state encodings and such that $j \neq j'$, their classes of $\sim^\sigma$ are distinct. The last property is the most involved, and to avoid making $|\phi_\mathcal{M}|$ exponential in n, we observe that $j \neq j'$ iff some binary digit is distinct:

$$\bigwedge_{d=1}^{n} \bigwedge_{b=0}^{1} \mathtt{G}\Big(b_d \Rightarrow {\downarrow_1} \, \mathtt{X} \, \neg\big(\neg({\uparrow_1}\!\sim \wedge\, \delta_d) \, \mathtt{U} \, ({\uparrow_1}\!\sim \wedge\, (1-b)_d)\big)\Big)$$

We have $|\tilde{\Sigma}| = O(n)$ and $|\phi_\mathcal{M}| = O(n^3 \log n)$. □

5 Refinement

The result in this section is that the problem of whether, given safety sentences ϕ and ϕ' of $\mathrm{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U})$, $\phi \Rightarrow \phi'$ is valid, is decidable but not primitive recursive. If for either ϕ or ϕ', the safety restriction is dropped, $\mathtt{F}^{-1}$ is allowed, or one more register is allowed, we have undecidability for the following reasons. Unsatisfiability of ϕ and validity of ϕ' are special cases of the problem. By the remarks at the beginning of Section 4, satisfiability and nonvalidity for $\mathrm{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U})$, and satisfiability for the safety fragments of $\mathrm{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U}, \mathtt{F}^{-1})$ and $\mathrm{LTL}_2^\downarrow(\sim; \mathtt{X}, \mathtt{U})$, are Π_1^0-complete. By Theorem 1 and the proof of [8, Theorem 17], validity for the safety fragments of $\mathrm{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U}, \mathtt{F}^{-1})$ and $\mathrm{LTL}_2^\downarrow(\sim; \mathtt{X}, \mathtt{U})$ is also Π_1^0-complete.

Theorem 5. *Refinement for safety* $\mathit{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U})$ *is decidable and not prim. rec.*

Proof. For decidability, suppose ϕ and ϕ' are sentences of safety $\mathrm{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U})$ over an alphabet Σ. By Theorem 1, let $\mathcal{A}_\phi^\Sigma$ and $\mathcal{A}_{\neg\phi'}^\Sigma$ be equivalent alternating RA with 1 register and with all and no (respectively) locations accepting, computed in logarithmic space. By the proof of Theorem 2, an IPCANT $\mathcal{C}$ corresponding to the intersection of $\mathcal{A}_\phi^\Sigma$ and $\mathcal{A}_{\neg\phi'}^\Sigma$ is computable in polynomial space, such that its transition relation contains no cycles of ε transitions, and each transition from an accepting location is to an accepting location.

As in the proof of Theorem 3, it suffices to consider lazy transitions. Let $\langle q, v\rangle \leq \langle q', v'\rangle$ iff $q = q'$ and $v \leq v'$. States and lazy transitions of $\mathcal{C}$ form a well-structured transition system with strong downward compatibility [17] with respect to $\leq$. The set A of all accepting states from which $\mathcal{C}$ has an infinite sequence of lazy transitions is downward-closed, and membership of A is decidable by the proof of Theorem 3. It remains to observe that reachability of A from the initial state is decidable by the forward set-saturation algorithm [17].

By the proof of [8, Theorem 15], there is a reduction (in logarithmic space) to validity for safety $\mathrm{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U})$ from emptiness of Incrementing counter automata over finite words, which is not primitive recursive [8, Theorem 2]. □

6 Model Checking

Our final result is that model checking for safety $\mathrm{LTL}_1^\downarrow(\sim; \mathtt{X}, \mathtt{U})$ is decidable and not primitive recursive. The validity problem is a special case, so by the remarks at the beginning of Section 5, we have that dropping the safety restriction, adding $\mathtt{F}^{-1}$, or adding one more register, each produce undecidability.

Theorem 6. *Model checking for nondeterministic RA and safety $LTL_1^{\downarrow}(\sim; \mathtt{X}, \mathtt{U})$ is decidable and not primitive recursive.*

Proof. Similar to the proof of Theorem 5, by constructing an IPCANT corresponding to the intersection of a given nondeterministic RA and an alternating RA which is equivalent to a given sentence of safety $\mathrm{LTL}_1^{\downarrow}(\sim; \mathtt{X}, \mathtt{U})$, has 1 register, and has no locations accepting. □

7 Concluding Remarks

It would be interesting to investigate whether ideas in this paper can be used to show that satisfiability for safety MTL [12] is primitive recursive.

I am grateful to Stéphane Demri and James Worrell for helpful discussions.

References

1. Schnoebelen, P.: The complexity of temporal logic model checking. In: AiML'02. Volume 4 of Advances in Modal Logic., King's College Publications (2003) 393–436
2. Lisitsa, A., Potapov, I.: Temporal logic with predicate λ-abstraction. In: TIME, IEEE (2005) 147–155
3. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L., David, C.: Two-variable logic on words with data. In: LICS, IEEE (2006) 7–16
4. Bojańczyk, M., David, C., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. In: PODS, ACM (2006) 10–19
5. Ouaknine, J., Worrell, J.: On the decidability of Metric temporal logic. In: LICS, IEEE (2005) 188–197
6. French, T.: Quantified propositional temporal logic with repeating states. In: TIME-ICTL, IEEE (2003) 155–165
7. Demri, S., Lazić, R., Nowak, D.: On the freeze quantifier in constraint LTL: decidability and complexity. In: TIME, IEEE (2005) 113–121
8. Demri, S., Lazić, R.: LTL with the freeze quantifier and register automata. In: LICS, IEEE (2006) 17–26
9. Kaminski, M., Francez, N.: Finite-memory automata. TCS **134**(2) (1994) 329–363
10. Neven, F., Schwentick, T., Vianu, V.: Finite state machines for strings over infinite alphabets. ACM TOCL **5**(3) (2004) 403–435
11. Sistla, A.P.: Safety, liveness and fairness in temporal logic. Formal Aspects of Computing **6**(5) (1994) 495–512
12. Ouaknine, J., Worrell, J.: Safety Metric temporal logic is fully decidable. In: TACAS. Volume 3920 of LNCS., Springer (2006) 411–425
13. Odifreddi, P.: Classical Recursion Theory II. Elsevier (1999)
14. David, C.: Mots et données infinies. Master's thesis, LIAFA (2004)
15. Schnoebelen, P.: Deciding termination of ICMETs (2006) Personal communication.
16. Brzozowski, J.A., Leiss, E.L.: On equations for regular languages, finite automata, and sequential networks. TCS **10**(1) (1980) 19–35
17. Finkel, A., Schnoebelen, P.: Well-structured transitions systems everywhere! TCS **256**(1–2) (2001) 63–92

Branching Pushdown Tree Automata*

Rajeev Alur and Swarat Chaudhuri

University of Pennsylvania

Abstract. We observe that pushdown tree automata (PTAs) known in the literature cannot express combinations of branching and pushdown properties. This is because a PTA processes the children of a tree node in possibly different control states but with identical stacks. We propose *branching pushdown tree automata* (BPTAs) as a solution. In a BPTA, a push-move views its matching pops as an unbounded, unordered set of successor moves and can assert existential and universal requirements on them, just the way finite automata on unranked, unordered trees pass requirements to the children of a tree node. We show that BPTAs can express some natural properties and are more expressive than PTAs. Using a small-model theorem, we prove their emptiness problem to be decidable. The problem becomes undecidable, however, if push-moves are allowed to specify the *ordering* of matching pops.

1 Introduction

Regular languages of trees [1] have been studied extensively in the literature [10] and found a number of applications. Automata accepting such languages can reason about paths in a tree existentially ("a symbol a is seen along *some* path from the current node") and universally ("a is seen on *all* paths"). Concretely, while reading a node in a binary tree, a nondeterministic, top-down tree automaton can nondeterministically pick pairs of different states to be sent to the children of the current node. Such "branching" of the finite control permits tree automata to specify properties of trees such as: "every node labeled a has a descendant labeled b and another descendant labeled c."

Above the class of regular tree languages in the hierarchy of expressiveness lies the class of *context-free* tree languages [7,1]. Such languages are accepted by *nondeterministic pushdown tree automata* (PTAs) [3,9,8,4,6], which augment tree automata with pushdown stores. PTAs are expressively equivalent to context-free tree grammars [1,7], and their emptiness problem is in EXPTIME [11,6]. Their usual operational definition runs as follows: while reading a tree node, a PTA $\mathcal{A}$ assumes a *configuration* of the form (q, w), where q is a state and w is a stack. At any point, $\mathcal{A}$ may push or pop the stack, or it may fork copies to be sent to the children of the current node. The essence of the expressiveness of PTAs, however, lies in the fact that they allow information stored on the stack at a push-transition to be retrieved at "matching" pop-transitions arbitrarily far

* This research was partially supported by ARO URI award DAAD19-01-1-0473 and NSF award CCR-0306382.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 393–404, 2006.

away. Another way to view this is: a push-move in a PTA $\mathcal{A}$ can *constrain* its matching pops—for instance, it may require q' to be the only state reached via the latter. This is analogous to the way a transition in a tree automaton can constrain the automaton's state at the children of a tree node.

We note, however, that in existing definitions of PTAs, copies of the automaton forked by a branch-transition have *identical stacks*, even though they may differ in control state (in some definitions, the stacks may differ, but only in a bounded way). If a push stores γ on the stack, then at *every matching pop*, it is the same γ that must be popped. Thus, while γ may be used to require that every matching pop leads to state q', a push-move cannot assert properties such as: "there exists a unique matching pop leading to state q_1, and every other matching pop leads to state q_2." Intuitively, a PTA can only express universal (as opposed to universal *and* existential) matching requirements. On the other hand, tree automata can reason universally and existentially about the children of a tree node—for unranked trees, MSO-complete tree automata [5] have transitions asserting requirements such as: "there exists a unique child to which state q_1 is passed, and every remaining child gets state q_2." Thus, PTAs do not really combine the way tree automata specify branching properties with the way pushdown automata express matching requirements.

To see how this prevents PTAs from capturing the interplay of matching and tree branching, consider a basic pushdown language: that of words over brackets $[$, $]_1$ and $]_2$ where every bracket $[$ has a matching instance of $]_1$ or $]_2$. Now consider the language L of trees labeled by the above brackets where: (1) each node labeled $[$ has a single descendant labeled $]_1$ such that the path from the former to the latter is "matched," and (2) every other "matched descendant" is labeled $]_2$. A push-transition taken by a PTA at a node labeled $[$ (or within a bounded distance from it) can check that *all* matching brackets reachable from the point of push are of a certain type. However, no PTA can accept L.

In this paper, we introduce *branching pushdown tree automata* (BPTAs), a class of pushdown automata which run on trees but do not suffer from this shortcoming in expressiveness. A push-transition in a BPTA views the tree nodes reached via its matching pops as an unbounded, unordered set of successor nodes, and can assert existential and universal requirements on them. More precisely, a push-transition is of the form $q \to (q', \mathit{push}(\chi))$, where q is the source state, q' is the destination state, and χ, a *constraint* on the states reached by the matching pops, can demand a requirement such as: "state q_1 is reached through one matching pop, and the rest lead to q_2." Note how this is analogous to the way MSO-complete finite automata on unranked, unordered trees can assert requirements on the children of a tree node. Thus, the ability of tree automata to reason about tree branches is combined seamlessly with the power of pushdown automata to match brackets, letting BPTAs accept a "truly pushdown" class of tree languages. Note also that the language L may now be accepted easily. At nodes labeled $]_1$ and $]_2$, the BPTA $\mathcal{A}$ for L pops and moves respectively to states q_1 and q_2, then continues down the tree. At a node labeled $[$, $\mathcal{A}$ pushes, asserting the constraint χ on the matching pops, before it branches.

BPTAs enjoy closure properties similar to PTAs but are provably more expressive. The main technical result of this paper is an algorithm for their emptiness problem. The analogous problem for PTAs reduces to pushdown games [11]; however, such a reduction seems impossible in this case. Instead, we define a proof system that, for states q and constraints χ, derives facts such as "starting at state q with empty stack from the root of some tree, the automaton has a way to reach the leaves of that tree with empty stack, at states that together satisfy χ." Using a small-model theorem that states that a short proof exists for every proof in this system, we obtain a 3-EXPTIME algorithm for the emptiness problem. Intriguingly, checking emptiness becomes undecidable if we allow BPTAs to reason about the *order* among the matching pops of a push by allowing the constraints asserted at push-moves to be regular expressions.

The organization of this paper is as follows. In Sec. 2, we present some definitions we use in the rest of the paper. In Sec. 3, we formally define BPTAs, and in Sec. 4, we present our main decision procedure. We study the expressiveness of BPTAs in Sec. 5, and conclude with some discussion in Sec. 6.

2 Basics

Binary trees. Our models in this paper are *binary trees*. Let Σ be an input alphabet. A finite binary tree over Σ is a term given by the grammar $T := \perp \mid a(T, T)$, for $a \in \Sigma$. The tree $\perp$ is the *empty tree*, and the *root* of a tree $a(T_1, T_2)$ is the letter a. The i-th *leaf* of T is the i-th instance of $\perp$ in it (reading left-to-right). The *i-th composition* $(T \circ_i T')$ of T and T' is the term obtained by replacing the i-th leaf of T by T'.

Count constraints. Consider a finite set Q and a word $\alpha \in Q^+$. We denote the length of α by $|\alpha|$ and the i-th symbol in α by $\alpha(i)$. The *count* of $q \in Q$ in α is the number of times q occurs in α.

We will be interested in *count constraints* over Q. Such a constraint χ follows the grammar $\chi := (count(q) \geq k) \mid (count(q) = k) \mid \chi \wedge \chi$, for $k \in \mathbb{N}$. A word α satisfies χ (written as $\alpha \models \chi$) iff it satisfies each conjunct of form $(count(q) \geq k)$ or $(count(q) = k)$; the former holds iff the count N of q in α satisfies $N \geq k$, and the latter iff $N = k$. We assume our constraints to be in the simplest possible form, i.e. no two conjuncts refer to $count(q)$ for the same q.

Let us now construct an alphabet of *starred elements* $Q_* = \{q^* | q \in Q\}$. We will represent a count constraint χ over Q as a *multiset* $(\mathbf{U} \cup U)$, where $\mathbf{U}$ is a multiset over Q, and $U \subseteq Q_*$. For each $q \in Q$, let m_q be the number of occurrences of q in $\mathbf{U}$; if $q^* \in U$, then set $\tau_q = (count(q) \geq m_q)$, else set $\tau_q = (count(q) = m_q)$. Then we must have $\chi = \bigwedge_q \tau_q$. Intuitively, m_q copies of q in χ guarantees any word satisfying χ to have at least m_q occurrences of q; absence of q^* (q^* is read as "an unspecified number of q-s") guarantees that the constraint is an equality. For instance, $\chi = \{q_1, q_1^*, q_2\}$ represents the constraint $(count(q_1) \geq 1) \wedge (count(q_2) = 1) \wedge \bigwedge_{i \neq 1,2}(count(q_i) = 0)$.

In the sequel, we denote by $count(\chi, q)$ the number of times q appears in χ, and we define the *size* of χ to be $Size(\chi) = \sum_q count(\chi, q)$. Also, binary relations over constraints χ and χ' are to be interpreted as relations over the corresponding multisets. Now we define an "implied-by" relation $\preccurlyeq$ for count constraints. For constraints χ and χ', we define $\chi \preccurlyeq \chi'$ iff (1) for each $q \in Q$ such that $q^* \in \chi$, we have $count(\chi, q) \leq count(\chi', q)$, and (2) for each $q \in Q$ such that $q^* \notin \chi$, we have $count(\chi, q) = count(\chi', q)$. Clearly, if $\chi \preccurlyeq \chi'$, then for every word $\alpha \in Q^+$, we have $\alpha \models \chi' \Rightarrow \alpha \models \chi$.

For count constraints $\chi_1 = \mathbf{U}_1 \cup U_1$ and $\chi_2 = \mathbf{U}_2 \cup U_2$, where $\mathbf{U}_1, \mathbf{U}_2$ are multisets over Q and $U_1, U_2 \subseteq Q_*$, we define the *sum* $(\chi_1 + \chi_2)$ as $(\mathbf{U}_1 \cup \mathbf{U}_2) \cup (U_1 \cup U_2)$. Note that the union of $\mathbf{U}_1$ and $\mathbf{U}_2$ is a multiset union that duplicates states, whereas $U_1 \cup U_2$ is a simple set union. Likewise, for $q \in \mathbf{U}_1$, we define $(\chi_1 - \{q\})$ to be $(\mathbf{U}_1 \setminus \{q\}) \cup U_1$.

3 Branching Pushdown Tree Automata

Syntax and semantics. A *(top-down) branching pushdown tree automaton* (BPTA) is a tuple $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where Q is a finite set of states, Σ is an input alphabet, $q_0 \in Q$ is the initial state, and $F \subseteq Q$ is a set of final states. The transition relation δ consists of four kinds of transitions: (1) push-transitions $q \rightarrow (q', push(\chi))$, where $q, q' \in Q$ and χ is a count constraint over Q; (2) pop-transitions $q \rightarrow (q', pop)$, where $q, q' \in Q$; (3) swap-transitions $q \rightarrow q'$, where $q, q' \in Q$, and (4) branch-transitions $q \xrightarrow{a} (q_1, q_2)$, where $a \in \Sigma$ and $q_1, q_2 \in Q$.

Intuitively, while processing a binary tree, a BPTA is able to change its configuration using a push, pop or swap transition and process the *same tree* while in the new configuration. It may also read the root of the tree, fork two copies using a branch transition, and use them to inductively process the left and right subtrees of the present tree. We note that the assumption that the current input symbol is ignored during pushes, pops and swaps is only for simpler exposition, and does not limit expressiveness. Also, observe that the transitions of a BPTA do not manipulate a stack explicitly—indeed, we avoid the use of a stack altogether while defining runs of BPTAs. However, we will see that our definition can encode the usual stack-based semantics for pushdown automata. We will also see that pushdown tree automata (PTAs) can be encoded by BPTAs where for every constraint χ appearing in a push-transition, $Size(\chi) = 0$.

The semantics of a BPTA $\mathcal{A}$ is defined inductively via predicates $Run(q, \alpha, T)$, where $q \in Q$, α is a word over Q, and T is a binary tree. Intuitively, the predicate $Run(q, \alpha, T)$ is true iff the automaton has a run on the tree T, starting at state q with empty (implicit) stack and ending at the leaves of T with empty stack, such that α is obtained by reading from left to right the states of $\mathcal{A}$ at the leaves of T. Formally:

- $Run(q, q, \perp)$ is true;
- if $\mathcal{A}$ has a transition $q \rightarrow q'$, then $Run(q, q', \perp)$;
- if $T = a(\perp, \perp)$ and $\mathcal{A}$ has a transition $q \xrightarrow{a} (q_1, q_2)$, then $Run(q, q_1 q_2, T)$;

- assume that $Run(q', \alpha', T)$ and $\mathcal{A}$ has a transition $q \to (q', push(\chi))$. Then $Run(q, \alpha, T)$ holds if for some $\alpha \in Q^*$, we have: (1) $\alpha \models \chi$, and (2) there is a bijection $\mu : \{1, 2, \ldots, |\alpha|\} \to \{1, 2, \ldots, |\alpha|\}$ such that $\mathcal{A}$ has a transition $\alpha(i) \to (\alpha'(\mu(i)), pop)$ for all $1 \leq i \leq |\alpha|$;
- if $Run(q, \alpha, T)$ and $Run(q', \alpha', T')$, and $\alpha(i) = q'$, then $Run(q, \alpha'', T \circ_i T')$, where α'' is obtained by substituting $\alpha(i)$ by α'.

The BPTA $\mathcal{A}$ *accepts* a tree T if $Run(q_0, \alpha, T)$ for some word α over F. Informally, the acceptance condition requires the automaton to reach each leaf of T in a final state with an empty (implicit) stack. The *language* $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is the set of all trees it accepts.

Among the above, the fourth and the fifth clauses are the most interesting. The fourth clause captures matching—if $\mathcal{A}$ pushes to go from q to q', and there is an empty-stack-to-empty-stack run from q' to q'', then a pop from q'' to q''' matches the original push. The distinguishing feature of BPTAs is that the word obtained by reading the q''''s from left to right must now satisfy a count constraint χ. The fifth clause captures the way a run from q and ending, among others, at q', can be *composed* with a run from q'.

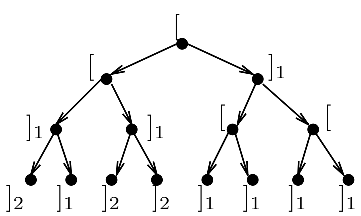


Fig. 1. A BPTA example

To see a language recognized by a BPTA, consider binary trees over the input alphabet $\Sigma = \{[,]_1,]_2\}$. Let nodes and paths in such trees have the natural definitions, and let brackets [be *matched* by brackets $]_1$ and $]_2$. Call a node x' in a tree a *matching node* of a node x if x is labeled [, x' is labeled $]_1$ or $]_2$, and there is a well-matched path (defined in the natural way) from x to x'. Now consider the language L of such trees where (1) every path from the root to a "leaf" $\perp$ (not including the leaf itself) is matched, and (2) every node labeled [has exactly two matching nodes labeled $]_1$, and every remaining matching node is labeled $]_2$. The tree in Fig. 1, for instance, belongs to L (the leaves have been omitted to keep the figure clean).

A BPTA $\mathcal{A}$ for L has states q, $q_{]_1}$, and $q_{]_2}$, the initial state being q. On reading a node labeled $]_1$ (similarly $]_2$), $\mathcal{A}$ pops and changes state to $q_{]_1}$ (or $q_{]_2}$). On reading a node labeled [, $\mathcal{A}$ pushes and sends the state q to the children of the current node, the count constraint in the push being: "state $q_{]_1}$ appears exactly twice, and $q_{]_2}$ occurs 0 or more times."[1] It is easy to see that $\mathcal{A}$ accepts L.

Pushdown tree automata. A *(top-down) pushdown tree automaton* (PTA) $\mathcal{P}$ has a finite state set H, an initial state h_0, a finite stack alphabet Γ, a set of final states, and transitions of the types $h \to (h_1, push(\gamma))$, $h \to (h_1, pop(\gamma))$, $h \to h_1$, and $h \xrightarrow{a} (h_1, h_2)$, where $h, h_1, h_2 \in H$, $\gamma \in \Gamma$, and a is an input symbol. A *configuration* of $\mathcal{P}$ is of the form (h, w), where $h \in H$, and $w \in \Gamma^*$ is

[1] While BPTAs, as defined, cannot push and branch in a "compound" transition, a move like this can be implemented using extra "book-keeping" states.

a *stack*. We define the semantics of $\mathcal{A}$ on an input tree T via predicates of the type $Accept((h, w), T)$, which intuitively means "$\mathcal{P}$ accepts T from configuration (h, w)," and is true if one of the following conditions holds:

- $w = \epsilon$, h is a final state, and $T = \perp$;
- there is a transition $h \rightarrow h'$ such that $Accept((h', w), T)$;
- $T = a(T_1, T_2)$, and for some transition $h \xrightarrow{a} (h_1, h_2)$, $Accept((h_1, w), T_1)$ and $Accept((h_2, w), T_2)$;
- there is a transition $h \rightarrow (h', push(\gamma))$ such that $Accept((h', \gamma.w), T)$;
- $w = \gamma.w'$, and for some transition $h \rightarrow (h', pop(\gamma))$, $Accept((h', w'), T)$.

The automaton $\mathcal{P}$ accepts a tree T if $Accept((h_0, \epsilon), T)$ holds; $\mathcal{L}(\mathcal{P})$ is the language of $\mathcal{P}$. Now, to see that BPTAs can encode PTAs, note that the only way a push-transition is different from a swap transition is that it *constrains* the "matching" pop transitions—if γ is pushed, then it is γ that must be popped at every matching pop. More precisely, construct from $\mathcal{P}$ a BPTA $\mathcal{A}$ with state set $H \cup (H \times \Gamma)$—intuitively, a pop in $\mathcal{A}$ to state (h, γ) simulates a move in $\mathcal{P}$ that pops γ and changes state to h. Every branch and swap transition in $\mathcal{P}$ is also a transition in $\mathcal{A}$; $\mathcal{A}$ also has extra swap-transitions $(h, \gamma) \rightarrow h$ for all $h \in H, \gamma \in \Gamma$. For every pop-transition $h \rightarrow (h', pop(\gamma))$ in $\mathcal{P}$, $\mathcal{A}$ has a transition $h \rightarrow ((h', \gamma), pop)$, and for every push $h \rightarrow (h', push(\gamma))$ in $\mathcal{P}$, $\mathcal{A}$ has a transition $h \rightarrow (h', push(\chi))$, where the count constraint $\chi = \bigwedge_h (count((h, \gamma) \geq 0) \wedge \bigwedge_{\forall h. q \neq (h,\gamma)} (count(q) = 0)$. It is not hard to see that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{P})$. Now, let a 0-BPTA be a BPTA where for every constraint χ appearing in a push-transition, we have $Size(\chi) = 0$. We note that $\mathcal{A}$ is a 0-BPTA. We can also show that for every 0-BPTA, there is an equivalent PTA. Then:

Theorem 1. *The class of languages accepted by PTAs equals the class of languages accepted by BPTAs where for every constraint χ appearing in a push-transition, we have $Size(\chi) = 0$.*

4 Emptiness

Now we present our main technical result: a decision procedure for the problem of checking, given a BPTA $\mathcal{A}$, whether $\mathcal{L}(\mathcal{A})$ is empty.

Consider a BPTA $\mathcal{A} = (Q, \Sigma, q_0, \delta, F)$, where $Q = \{q_1, q_2, \ldots, q_n\}$, and recall the predicates $Run(q, \alpha, T)$ defined in Sec. 3. We would like to prove such predicates inductively—however, since they are unboundedly many, we must quotient them in a finite way.

We do so using predicates of the form $\mathcal{F}(q, \chi)$, where $q \in Q$ and χ is a *count constraint* over Q. The predicate $\mathcal{F}(q, \chi)$ holds iff $Run(q, \alpha, T)$ holds for some tree T and some word $\alpha \in Q^+$ such that $\alpha \models \chi$. Then, by definition:

Lemma 1. *If $\chi \preccurlyeq \chi'$, then for all q, $\mathcal{F}(q, \chi') \Rightarrow \mathcal{F}(q, \chi)$.*

Lemma 2. *$\mathcal{L}(\mathcal{A}) \neq \emptyset$ iff $\mathcal{F}(q_0, \chi_F)$, where $\chi_F = \{q^* : q \in F\}$.*

A natural question is whether it suffices to consider only predicates $\mathcal{F}(q,\chi)$ where χ appears in a push-transition of $\mathcal{A}$. It turns out that it does not. Consider a BPTA $\mathcal{A}$ that has, among others, a push-transition $q \rightarrow (q', push(\psi))$, where $\psi = \{p_1^*, p_2, p_2^*\}$, and pop-transitions $q'' \rightarrow (p_1, pop)$ and $q'' \rightarrow (p_2, pop)$. Now suppose the constraint $\chi = \{p_1, p_1^*, p_2^*\}$ appears in a different push-transition, and that we want to prove $\mathcal{F}(q,\chi)$. We note that to use the push-transition involving ψ in such a proof, we need to prove the stronger predicate $\mathcal{F}(q,\chi'')$, where $\chi'' = \chi + \psi = \{p_1, p_1^*, p_2, p_2^*\}$. If we are to use this push along with the pop-transitions from q'', we also need to prove $\mathcal{F}(q',\chi')$, where $\chi' = \{q'', q'', q''^*\}$. However, there is no reason why χ' must appear in a transition of $\mathcal{A}$.

Hence we define a proof system $\mathbb{F}$ for facts $\mathcal{F}(q,\chi)$. The system derives predicates $\mathsf{F}(q,\chi)$ (designed to be the syntactic analog of $\mathcal{F}(q,\chi)$), and uses the rules:

1. $\dfrac{}{\mathsf{F}(q,\chi)}$, for $\chi \preccurlyeq \{q\}$ (Base)
2. $\dfrac{\mathsf{F}(q,\chi) \qquad \mathsf{F}(q',\chi')}{\mathsf{F}(q,\chi'')}$ (Compose),
   if q' is in χ and $\chi'' \preccurlyeq \chi + \chi' - \{q'\}$.
3. $\dfrac{}{\mathsf{F}(q,\chi)}$ (Swap),
   if $\mathcal{A}$ has a transition $q \rightarrow q'$ and $\chi \preccurlyeq \{q'\}$.
4. $\dfrac{}{\mathsf{F}(q,\chi)}$ (Branch),
   if $\mathcal{A}$ has a transition $q \xrightarrow{a} (q_1, q_2)$ for some a such that $\chi \preccurlyeq \{q_1, q_2\}$.
5. $\dfrac{\mathsf{F}(q',\chi')}{\mathsf{F}(q,\chi)}$ (Summarize),
   if there are count constraints χ'' and ψ such that: (1) $\chi \preccurlyeq \chi''$, (2) $\psi \preccurlyeq \chi''$, (3) $\mathcal{A}$ has a push-transition $q \rightarrow (q', push(\psi))$, and (4) there is a relation $\nu \subseteq \chi' \times \chi''$ such that:
   (a) for every $v \in \chi''$, there is some $u \in \chi'$ such that $\nu(u,v)$
   (b) for each $u \in \chi'$ that is a state of $\mathcal{A}$, we have a unique $v \in \chi''$ such that $\nu(u,v)$, v is a state of $\mathcal{A}$, and $\mathcal{A}$ has a transition $u \rightarrow (v, pop)$;
   (c) for each u of form q''^*, for $q'' \in Q$, in χ', every $v \in \chi''$ such that $\nu(u,v)$ must satisfy: (i) v is of form q'''^* for some $q''' \in Q$, and (ii) $\mathcal{A}$ has a transition $q'' \rightarrow (q''', pop)$.

Here, the rule Base may be used to establish that $\mathcal{F}(q,\{q\})$ is true. In addition, this rule can prove a "weaker" fact such as $\mathcal{F}(q,\{q,q^*\})$, implied by $\mathcal{F}(q,\{q\})$ according to Lemma 2. Generally, if any of our rules can derive a fact, then it can also derive every weaker fact.

We will explain why the rule Compose is sound to demonstrate its purpose. Suppose we have $\mathsf{F}(q,\chi)$ and $\mathsf{F}(q',\chi')$, for $q' \in \chi$. Inductively, we have $Run(q,\alpha,T)$ for some α, T such that $\alpha \models \chi$; likewise, we have $Run(q',\alpha',T')$ for some α', T' such that $\alpha' \models \chi'$. Since $q' = \alpha(i)$ for some i, we have, by the semantics of $\mathcal{A}$, $Run(q,\alpha'',T \circ_i T')$, where α'' is obtained by replacing the i-th letter of α by α'. As $\alpha'' \models \chi + \chi' - \{q'\}$, we can soundly derive any goal weaker than $\mathsf{F}(q, \chi + \chi' - \{q'\})$.

Rules Swap and Branch capture the semantics of the swap and branch transitions of $\mathcal{A}$. We will explain the rule Summarize via an example (Fig. 2). Suppose we want to derive $\mathsf{F}(q, \chi)$, where $\chi = \{p_1, p_1^*, p_2\}$. Let $\mathcal{A}$ have a push-transition $q \rightarrow (q', \mathit{push}(\psi))$ that, matched by some pop-transitions and combined with the true predicate $\mathcal{F}(q', \chi')$, proves $\mathcal{F}(q, \chi'')$ for some χ'' satisfying $\chi \preccurlyeq \chi''$. We must have $\psi \preccurlyeq \chi''$; also, there must exist appropriate pop-transitions from χ' to χ''. To be concrete, let $\chi'' = \{p_1, p_1, p_1, p_1^*, p_2\}$, and $\psi = \{p_1, p_1, p_1^*, p_2\}$, and suppose $\mathcal{A}$ has pop transitions $q \rightarrow (p_1, \mathit{pop})$, $q' \rightarrow (p_1, \mathit{pop})$, and $q' \rightarrow (p_2, \mathit{pop})$. Now, every instance of p_1 (or p_2) in χ'' guarantees one copy of p_1 (p_2) reached in a run of $\mathcal{A}$, and must be derived via a pop from a copy of q or q' in χ'. The element p_1^* stands for "an unspecified number (zero or more) of p_1's," and must be derived from "an unspecified number of states that, via a pop, may lead to p_1." Thus, we may set $\chi' = \{q, q, q', q^*, q', q'^*\}$ (as in the figure) or $\chi' = \{q, q, q, q'^*, q'\}$, but not, say, $\chi' = \{q, q, q, q^*, q\}$. Now, the relation $\nu \subseteq \chi' \times \chi''$ collects the pairs (u, v) such that v is derived from u.

Let us write $\vdash_{\mathbb{F}} \mathsf{F}(q, \chi)$ if $\mathsf{F}(q, \chi)$ is derivable in $\mathbb{F}$. We can prove that:

Lemma 3. *$\mathcal{F}(q, \chi)$ iff $\vdash_{\mathbb{F}} \mathsf{F}(q, \chi)$.*

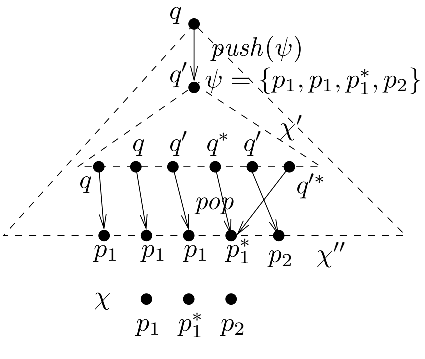


Fig. 2. The rule Summarize

Now take a proof tree for $\mathsf{F}(q_0, \chi_F)$, where χ_F is as in Lemma 2. We show that for every such tree, there is a proof for the same predicate involving a small number of predicates. Consider a path in this tree from the root (the target predicate $\mathsf{F}(q_0, \chi_F)$) to a leaf (a predicate derived without a premise). For predicates P and P' that lie on such a path, let us write $\mathsf{P} \lhd \mathsf{P}'$ if P is derived using P' in one step. We write $\mathsf{P} \lhd^+ \mathsf{P}'$ if P is obtained via a positive number of derivations from P'. Call a proof tree $\mathcal{S}$ *minimal* if it cannot be further reduced by any of the following two operations: (1) replace the proof for a predicate P in $\mathcal{S}$ by a proof tree with fewer vertices, and (2) if $\mathsf{F}(q, \chi) \lhd \mathsf{F}(q', \chi')$ in $\mathcal{S}$, then replace $\mathsf{F}(q', \chi')$ by a predicate $\mathsf{F}(q', \chi'')$, where $\chi'' \preccurlyeq \chi'$ and $\chi'' \neq \chi'$, such that $\mathsf{F}(q, \chi)$ can be derived from $\mathsf{F}(q', \chi'')$ (and the other predicates used to derive $\mathsf{F}(q, \chi)$ in $\mathcal{S}$). Note that in the above, a proof for $\mathsf{F}(q', \chi'')$ follows directly from the proof for $\mathsf{F}(q', \chi')$).

Let $\mathcal{S}$ be a minimal proof tree. We note that if $\chi \preccurlyeq \chi'$ for some χ and χ', and $\mathsf{F}(q, \chi) \lhd^+ \mathsf{F}(q, \chi')$ for some q in Q, then by Lemma 1, we can compact $\mathcal{S}$ by replacing the proof of $\mathsf{F}(q, \chi)$ by the (stronger) proof for $\mathsf{F}(q, \chi')$. Since $\mathcal{S}$ is minimal, this is a contradiction, so that:

Lemma 4. *In a minimal proof tree, we cannot have $\mathsf{F}(q, \chi) \lhd^+ \mathsf{F}(q, \chi')$ if $\chi \preccurlyeq \chi'$.*

Now note that if $\mathsf{F}(q,\chi) \lhd \mathsf{F}(q',\chi')$ in a minimal proof tree, then we can have $\mathit{Size}(\chi) < \mathit{Size}(\chi')$ only if the rule Summarize is used for this derivation. Now consider a state p such that $p^* \in \chi$, and let $c_{\max} = \max_\psi \max_q (\mathit{count}(\psi,q))$ be the maximum count of a state in a count constraint that appears in a push-transition of $\mathcal{A}$. Because we must have $\psi \preccurlyeq \chi''$, it may not suffice to have $\mathit{count}(\chi'',p) = \mathit{count}(\chi,p)$, but the number of extra copies of p that we may need to add is at most $c_{\max}$ (we may also have to add elements of the form q^*, but recall that they do not figure in the size of a constraint). At the same time, for states p' for which $p'^* \notin \chi$, we cannot have $\mathit{count}(\chi,p') > c_{\max}$—this is because $\mathit{count}(\chi,p') = \mathit{count}(\chi'',p') = \mathit{count}(\psi,p')$, which contradicts the definition of $c_{\max}$. Setting $k = \theta(nc_{\max})$ for the rest of this section, we have $\mathit{Size}(\chi') \leq \mathit{Size}(\chi) + k$. Then:

Lemma 5. *If* $\mathsf{F}(q,\chi) \lhd \mathsf{F}(q',\chi')$ *in a minimal proof, then* $\mathit{Size}(\chi') \leq \mathit{Size}(\chi) + k$.

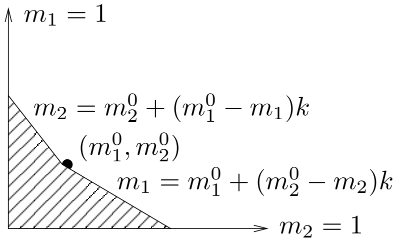

$m_1 = 1$
$m_2 = m_2^0 + (m_1^0 - m_1)k$
(m_1^0, m_2^0)
$m_1 = m_1^0 + (m_2^0 - m_2)k$
$m_2 = 1$

Fig. 3. Bound on constraint size

Now we abstract the problem further. From now on, only consider count constraints χ where $q \in \chi \Rightarrow q^* \in \chi$. Call a sequence of such count constraints $\varphi = \chi_1\chi_2\ldots\chi_l$ a *proof witness* if it satisfies the conditions: (1) for all i, $\mathit{Size}(\chi_{i+1}) \leq \mathit{Size}(\chi_i) + k$, (2) if $j > i$, then we cannot have $\chi_i \preccurlyeq \chi_j$, and (3) $\chi_1 = \chi_F$, where χ_F is as in Lemma 2. Let us denote by $\lambda(\varphi)$ the maximum size of a constraint in a proof witness φ. We ask the question: what is the maximum value of $\lambda(\varphi)$ over all proof witnesses φ? The answer is an upper bound on the maximum size of a count constraint χ appearing in a predicate $\mathsf{F}(q,\chi)$ in a minimal proof tree.

Define the *basis* of a count constraint χ as the set of states q such that $q^* \in \chi$. The total number of bases is bounded by 2^n. Using the facts that the ordering $\preccurlyeq$ only relates count constraints over the same basis and that we assume nothing about the specific counts of states in a constraint, we observe:

Lemma 6. *For any proof witness* $\varphi = \chi_1\chi_2\ldots\chi_l$*, there are* 2^n *sequences of count constraints* φ_1*,* φ_2*,...,*φ_{2^n}*, such that (1) each constraint appearing in a particular* φ_i *has the same basis, (2) constraints in different* φ_i*'s have different bases, and (3) the concentenation* φ' *of the* φ_i*'s is a proof witness satisfying* $\lambda(\varphi) \leq \lambda(\varphi')$.

Call a proof witness *contiguous* if it may be split into sub-witnesses over particular bases in the above way. To find $\lambda(\varphi')$ for a contiguous proof witness $\varphi' = \varphi_1\varphi_2\ldots\varphi_{2^n}$, we consider a sub-witness $\varphi_i = \chi_1\chi_2\ldots\chi_l$, such that $\mathit{count}(\chi_1,q) = m$ for all q in the common basis of constraints in φ_i (in general, the state q_i can have a count m_i, but we could set $m = \max_i m_i$ without decreasing $\lambda(\varphi)$). We will find a bound $\pi(m,n,k)$ on $\lambda(\varphi_i)$ under this assumption. First, note that for $i_1 > i_2$, we cannot have $count(\chi_{i_1}, q) > count(\chi_{i_2}, q)$ for *all* q in the basis of φ_i. Then, starting from χ_1, if we decrease the count of one of the states to 1, then the other counts can grow at most to $(m + (m-1)k) = O(mk)$ (the situation is illustrated in Fig. 3-(b) for basis size 2; here, allowed pairs (m_1, m_2) of counts are depicted as points in a 2-dimensional space, and can only lie in its shaded part—(m_1^0, m_2^0) is the "initial" point). In this way we can show that $\pi(m, n, k) \leq \pi(O(mk), n-1, k)$, and using this inequality, that $\lambda(\varphi_i) = O(m^n k^n)$. Now note that in φ_{i+1}, the first constraint has the form $(m', m', \ldots, m')$, where $m' = O(m^n k^n)$, so that $\lambda(\varphi_{i+1}) = O(m^{n^2} k^{n^2+n})$. Also, in the first constraint in φ_1, the count of each state is 0. From all this and using induction, we obtain that $\lambda(\varphi) = O(k^{n^{2^n}})$.

Now note that the total number of multisets over a basis of size n where each element can have at most r copies is r^n. Therefore, the total number of predicates $\mathsf{F}(q, \chi)$ is $O(k^{n^{2^n}})$. Since every derivation step in $\mathbb{F}$ derives at least one new predicate, we have:

Theorem 2. *The emptiness problem of BPTAs is in 3-EXPTIME.*

5 Expressiveness

Basic properties. In this section, we study the expressiveness of BPTAs further. First, note that on word models, a push in a run of a BPTA has a single matching pop, so that the count constraints in push-transitions applicable in this setting can be simplified to: "one of the states in $Q' \subseteq Q$ appears once, and the other states do not occur." This can be encoded by nondeterministic pushdown word automata, proving (along with Theorem 1) that BPTAs on words accept precisely the class of context-free languages.

As for closure properties of BPTAs, closure under union is trivial. Some "hardness" results follow from Theorem 1 and results for pushdown automata:

Theorem 3. *BPTAs are closed under union, but not under intersection or complementation. The problems of checking the emptiness of (1) the complement of a BPTA and (2) the intersection of two BPTAs are undecidable.*

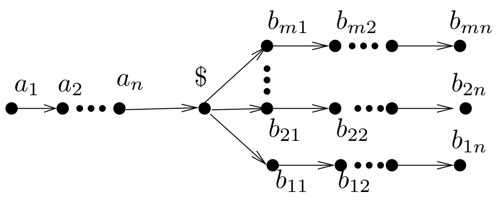


Fig. 4. Expressiveness of BPTAs

We show that BPTAs are more expressive than PTAs by considering trees as in Fig. 4 (the leaves have been omitted). Here, the input alphabet is $\Sigma = \{0, 1, \$\}$, and the symbols a_i, b_{ij} are in Σ for all i, j (while these trees are not binary, we can always encode them by such). Now let L be the language of trees of the above form where for all $i \leq n$, there is exactly one $k \leq m$ such that $a_{n-i+1} = b_{ki}$. This language is recognized by a BPTA that has states

q_0 and q_1 (along with a couple of other states needed for "book-keeping") corresponding to the input symbols 0 and 1. While reading each a_i, it executes a push-transition that enforces the following count constraint χ on its matching pops: "state q_{a_i} appears exactly once, and state q_j, where $j \neq a_i \in \{0, 1\}$ can appear an unspecified number of times." On reading a symbol b_{ij}, the BPTA executes a pop-transition to the state $q_{b_{ij}}$.

To see why L cannot be recognized by a PTA $\mathcal{M}$ with N states, take a tree as above where $n = m > N$. In any run, $\mathcal{M}$ must enter two branches of the tree in the same configuration. Then we can replace one of these branches with the other to get an accepting run on a tree not in L. This leads to:

Theorem 4. *There is a BPTA $\mathcal{A}$ such that no PTA recognizes $\mathcal{L}(\mathcal{A})$.*

Alternation. One may wonder if BPTAs can be simulated by *alternating* pushdown tree automata (APTAs), which can fork copies during a run and require that all forked copies accept the input tree. Such automata have undecidable membership and emptiness problems and can accept languages not recognizable by BPTAs. For instance, the non-context free word language $L = \{a^i b^i c^i : i \geq 1\}$, clearly not accepted by a BPTA, can be accepted by an APTA [6].

However, alternation does not appear to be the source of expressiveness of BPTAs. Consider the language L of trees as in Fig. 4 where there is a $j \leq n$ such that for all $i \leq j$, there is a branch k such that $a_{n-j+1} = b_{kj}$ and $a_{n-i+1} = b_{ki}$. An APTA $\mathcal{M}$ running on such trees cannot track the universal quantifier over i just by forking copies. Such copies would run independently and not agree on the value of j. We conjecture that L cannot be accepted by an APTA. On the contrary, consider a BPTA $\mathcal{A}$ that has a pair of states $q_\sigma, q_\sigma^\#$ for each $\sigma \in \Sigma$, and pushes on the a's and pops on the b's. At every a_i preceding some nondeterministically guessed a_j, $\mathcal{A}$ pushes and asserts that, among the states reached via the matching pops, "q_{a_i} appears at least once." At a_j, $\mathcal{A}$ demands that "$q_{a_j}^\#$ occurs once or more" among the states reached by the matching pops. While popping along the k-th branch of b's, $\mathcal{A}$ has, in the beginning, the option to move to a state $q_\sigma^\#$ at any point. If it does so on a symbol b_{kl}, then it checks that $\sigma = b_{kl}$. Now it waits to move to a state $q_{\sigma'}$. If it does so on a symbol b_{kp}, then it checks that $\sigma' = b_{kp}$. We can show that $\mathcal{A}$ accepts a tree if and only if it belongs to L.

Regular expressions instead of count constraints? While a count constraint χ in a push-transition in a BPTA $\mathcal{A}$ can reason about *state counts* in the multiset of states reached via the pops matching the push, it cannot *order* them by the position of the leaves they reach. A way to let BPTAs reason about the order of matching pops would be to let χ be a *regular expression.* The semantics for push-transitions is the obvious one; pop, swap and branch transitions stay the same.

Such automata can trivially encode BPTAs; unfortunately, their emptiness problem is undecidable (we omit the proof). Evidently, the expressiveness of BPTAs is quite close to the maximum permitted by decidability contraints.

6 Conclusions

In this paper, we introduced BPTAs as a new automaton model for pushdown tree languages. Unlike pushdown tree automata studied in the literature, BPTAs allow path quantifiers to be combined with pushdown properties satisfied along a path. We established that BPTAs are strictly more expressive than classical PTAs and presented a decision procedure for their emptiness problem.

There is an intriguing connection between our decidability result and known results [2] for transition systems equipped with well-founded quasi-orders (wqo). Using Lemma 6, we can establish that the relation $\preccurlyeq$ defines a wqo on a transition system whose states are predicates of the form $\mathsf{F}(q, \chi)$. We can then pose the emptiness question for BPTAs as an *alternating coverability problem* on this transition system, which can then be proved decidable by extending existing decidability proofs for coverability in such systems.

Several questions are left open. First, we are not convinced that the upper bound for our decision procedure is tight, and it is possible that an entirely new approach would yield a better upper bound. Secondly, an extension of context-free tree grammars that is equivalent to BPTAs would be interesting to study. Finally, this paper exclusively deals with automata on finite trees, and a generalization to infinite trees and infinitary acceptance conditions would be of interest.

Acknowledgement. We thank P. Madhusudan for valuable discussions.

References

1. H. Comon, M. Dauchet, R. Gilleron, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications. Draft, 2002.
2. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theoretical Computer Science*, 256(1-2):63–92, 2001.
3. I. Guessarian. Pushdown tree automata. *Math. Systems Theory*, 16(4):237–263, 1983.
4. D. Harel and D. Raz. Deciding emptiness for stack automata on infinite trees. *Information and Computation*, 113(2):278–299, 1994.
5. D. Janin and I. Walukiewicz. On the expressive completeness of the propositional mu-calculus with respect to monadic second order logic. In *CONCUR 1996*, LNCS 1119, pages 263–277. Springer-Verlag, 1996.
6. O. Kupferman, N. Piterman, and M.Y. Vardi. Pushdown specifications. In *LPAR 2002*, LNCS 2514, pages 262–277. Springer, 2002.
7. W. C. Rounds. Mappings and grammars on trees. *Mathematical Systems Theory*, 4(3):257–287, 1970.
8. A. Saoudi. Pushdown automata on infinite trees and nondeterministic context-free programs. *International Journal of Foundations of Comp. Sci.*, 3(1):21–39, 1992.
9. K. M. Schimpf and J. H. Gallier. Tree pushdown automata. *Journal of Computer and System Sciences*, 30(1):25–40, 1985.
10. W. Thomas. Languages, automata, and logic. *Handbook of Formal Language Theory*, III:389–455, 1997.
11. I. Walukiewicz. Pushdown processes: Games and model-checking. *Information and Computation*, 164(2):234–263, 2001.

Validity Checking for Finite Automata over Linear Arithmetic Constraints*

Gary Wassermann and Zhendong Su

Department of Computer Science, University of California, Davis
{wassermg, su}@cs.ucdavis.edu

Abstract. Decision procedures underlie many program analysis problems. Traditional program analysis algorithms attempt to prove some property about a single, statically-defined program by generating a single constraint. Accordingly, traditional decision procedures take single constraints as input. Extending these traditional program analysis algorithms to reason about potentially infinite languages of programs (as generated by a given metaprogram) requires a new class of decision procedures that reason about languages of constraints. This paper introduces the parameterized class of validity checking problems that take as input a language generator $\mathcal{A}$. The parameters are: (1) the language formalism for $\mathcal{A}$, (2) the theory under which each string in the language of $\mathcal{A}$ is interpretted, and (3) the quantification (existential/universal) of the constraints in the language to which the validity property applies. We introduce such decision problems by presenting an algorithm that decides whether a given finite state automaton $\mathcal{A}$ generates any valid linear arithmetic constraints.

1 Introduction

Many program analysis and formal verification problems reduce to validity or satisfiability checking over some logical theories. Consequently, significant effort has been devoted to designing efficient decision procedures for these theories. Traditional program analysis problems address individual programs, so the decision procedures that underlie program analysis algorithms take a single constraint φ. Extending program analysis problems to address potentially infinite languages of programs (as generated by a metaprogram) requires decision procedures that take languages of constraints. We introduce the study of such decision procedures in this paper. The input to decision procedures over languages of constraints is a language generator $\mathcal{A}$, where each string in the language of $\mathcal{A}$ is a constraint in a given theory. The problem such procedures address is: Does there exist a valid constraint in the language of $\mathcal{A}$, or alternatively, are all constraints in the language of $\mathcal{A}$ valid?

As an example application, consider a web application that takes user input (e.g., a username and password) and generates a query to a backend database (e.g., a banking system) to authenticate the user. Errors in the application may allow a malicious user to send specifically crafted input to cause the application to generate a query with a tautology as its conditional clause. This is one example of a widespread security vulnerability

* This research was supported in part by NSF CAREER Grant No. 0546844 and a generous gift from Intel. The information presented here does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 405–416, 2006.

$$\begin{array}{l} G = (\ \{\vee, \wedge, \neg, (,), -, +, =, \neq, >, \leq, <, \geq, \} \cup \Re \cup V, \\ \quad \{\mathsf{bE}, \mathsf{bT}, \mathsf{bF}, \mathsf{bS}, \mathsf{pred}, \mathsf{aE}, \mathsf{aT}\}, \ P_G, \ \mathsf{bE}) \end{array}$$

$$P_G = \left\{ \begin{array}{lcllcl} \mathsf{bE} & ::= & \mathsf{bE} \vee \mathsf{bT} \ \mid \ \mathsf{bT} & \mathsf{aE} & ::= & \mathsf{aE} + \mathsf{aT} \ \mid \ \mathsf{aT} \\ \mathsf{bT} & ::= & \mathsf{bT} \wedge \mathsf{bF} \ \mid \ \mathsf{bF} & \mathsf{aT} & ::= & V \ \mid \ -V \ \mid \ \Re \\ \mathsf{bF} & ::= & \neg\, \mathsf{bS} \ \mid \ \mathsf{bS} & \mathsf{cmp} & ::= & = \ \mid \ \neq \ \mid \ > \\ \mathsf{bS} & ::= & (\, \mathsf{bE}\,) \ \mid \ \mathsf{pred} & & \mid & \geq \ \mid \ < \ \mid \ \leq \\ \mathsf{pred} & ::= & \mathsf{aE}\ \mathsf{cmp}\ \mathsf{aE} & & & \end{array} \right\}$$

Fig. 1. Grammar G for linear arithmetic constraints, where V is a set of variables

known as *database command injection* [1]. These vulnerabilities can be discovered statically by constructing a language generator $\mathcal{A}$ to conservatively characterize the set of database queries that the application may generate [2]. The verification problem then reduces to checking whether $\mathcal{A}$ accepts any tautologies.

We denote this class of problems parametrically as $\text{VALID}_{\Pi,\Phi,K}$. The first parameter, Π, is the formalism for describing the language generator $\mathcal{A}$ that $\text{VALID}_{\Pi,\Phi,K}$ takes as input. The second parameter, Φ, is the theory under which each string in $\mathcal{L}(\mathcal{A})$ (i.e., the language of $\mathcal{A}$) is to be interpreted. The third parameter, $K \in \{\exists, \forall\}$, specifies whether the goal is to find whether any ($K = \exists$) or all ($K = \forall$) constraints in $\mathcal{L}(\mathcal{A})$ are tautologies. This paper introduces such decision problems by presenting an algorithm for $\text{VALID}_{\text{FSA,LA},\exists}$, where "FSA" is short for "Finite State Automaton," and "LA" is short for "Linear Arithmetic." In practice, FSAs are sufficient for modeling web applications as query constructors [2].

The challenge of $\text{VALID}_{\Pi,\Phi,K}$ for any non-trivial Π is that $\mathcal{L}(\mathcal{A})$ may be infinite, so naively enumerating $\mathcal{L}(\mathcal{A})$ and checking each constraint will not yield a decision procedure. Instead, the algorithm must exploit the finiteness of the representation of $\mathcal{A}$.

The rest of the paper is structured as follows. Section 2 presents the $\text{VALID}_{\text{FSA,LA},\exists}$ problem more precisely and defines *arithmetic loops* and *logical loops*, which represent the main challenges of the problem. Sections 3 and 4 address arithmetic and logical loops respectively. Section 5 surveys related work, and Sect. 6 concludes.

2 Overview

This section first defines the parameters for $\text{VALID}_{\text{FSA,LA},\exists}$ and makes some general observations, and then sets up the high-level structure of the algorithm.

2.1 The $\text{VALID}_{\text{FSA,LA},\exists}$ Problem

Finite state automata (FSAs) are defined by a five-tuple, $(Q, \Sigma, \delta, q_0, q_f)$, where Q is a set of states, Σ is the alphabet of terminals from the input language, $\delta \subseteq Q \times \Sigma \times Q$ is a transition relation, $q_0 \in Q$ is a start state, and $q_f \in Q$ is a final state. The semantics of FSAs is standard. The grammar G in Fig. 1 defines the syntax for linear arithmetic constraints. Again, the semantics of the language is standard, and the grammar rules reflect the operator precedence. Because each $s \in \mathcal{L}(\mathcal{A})$ must be interpreted as a linear arithmetic constraint, for $\mathcal{A}$ to be a valid input to $\text{VALID}_{\text{FSA,LA},\exists}$, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(G)$. For the sake of compactness and for certain steps in our algorithm, the transition relation will sometimes be presented as $\delta \subseteq Q \times \Sigma^* \times Q$.

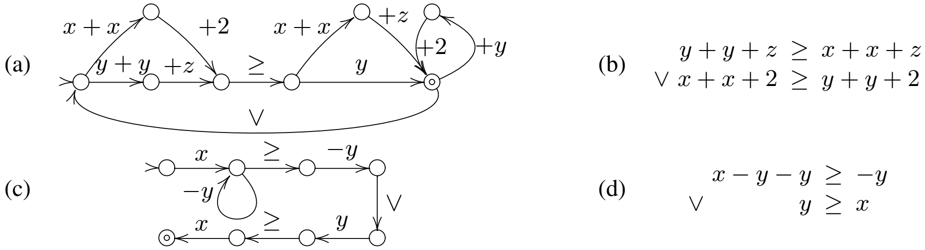


Fig. 2. Two example FSAs

$$\text{let}\quad a_{ik} = (q_i, a, q_k) \quad b_{ij} = (q_i, b, q_j) \quad c_{jk} = (q_j, c, q_k)$$
$$\begin{array}{lcl} a \rightarrow b\ c \in P_G & & \delta_N = \delta_N \cup \{a_{ik}\} \\ b_{ij}, c_{jk} \in \delta_T \cup \delta_N & \Rightarrow & \delta_R(a_{ik}) = \delta_R(a_{ik}) \cup \{\{b_{ij}, c_{jk}\}\} \end{array}$$

Fig. 3. CFL-reachability algorithm—the cases for *rhs*'s of lengths other than 2 are analogous

We select Φ = "LA" to explore because it is broadly applicable, and the general problem of validity checking for integer arithmetic constraints is undecidable (due to the undecidability of Diophantine equations [3]). Although multiplication by a constant is within the theory of linear arithmetic, we forbid '×' from appearing in Σ. If we allowed, for example, an FSA to have a loop over "×2," we would characterize the multiplication as "$\times 2^n$," and exponentiation with variables is difficult to reason about.

A few concrete examples of inputs to $\text{V\small{ALID}}_{\text{FSA,LA},\exists}$ help to illustrate the significance of the finite representation and the challenges in handling it. Consider, for example, the FSA shown in Fig. 2a. Because of cycles in the automaton, it accepts an infinite language. By considering single passes through each of its cycles, we discover the tautology shown in Fig. 2b. However, a single pass through a cycle is not sufficient to discover possible tautologies in general. For example, two passes through the cycle in the FSA shown in Fig. 2c are needed to discover the tautology in Fig. 2d.

Our algorithm for validity checking of automata uses a combination of automaton transformations and a theorem that bounds the number of constraints needed for a tautology. It generates validity queries in the theory of first-order arithmetic and sends them to a first-order arithmetic decision procedure [4]. If the FSA accepts some tautology, at least one of the finite number of first-order arithmetic queries must be a tautology.

2.2 Definitions and Setup

Our algorithm for the $\text{V\small{ALID}}_{\text{FSA,LA},\exists}$ problem uses a modified version of context free language (CFL) reachability to create abstractions of the input FSA for use at certain steps. This CFL-reachability algorithm takes as input a context free grammar $G = (N, \Sigma, P_G, S)$ and an FSA $\mathcal{A} = (Q, \Sigma, \delta, q_0, q_F)$, and produces an augmented FSA $\mathcal{A}' = (Q, \Sigma \cup N, \delta_T \cup \delta_N, \delta_R, q_0, q_F)$ where δ_T and δ_N are sets of *terminal transitions* and *non-terminal transitions* (transitions labeled with terminals and non-terminals from G) respectively, and $\delta_R : \delta_N \rightarrow \mathcal{P}(\mathcal{P}(\delta_N \cup \delta_T))$ is the set of *reference transitions*. The transitions in $\mathcal{A}'$ are defined by $\delta_T = \delta$ plus the minimal solution to the constraint shown in Fig. 3. The standard CFL-reachability algorithm does not include reference transitions [5]. Figure 4 depicts an FSA produced by CFL-reachability, showing terminal transitions as solid, nonterminal transitions as dashed, and reference transitions as dotted, assuming that $A \rightarrow B\ C \in P_G$. For $t \in \delta_N$, we write $t \rightsquigarrow t'$ if $t \in s_t \in \delta_R(t)$ for some s_t; let '$\rightsquigarrow^*$' denote the reflexive, transitive closures of '$\rightsquigarrow$,' and let $\delta^*_R(t) = \bigcup_{s_t \in \delta_R(t)}(s_t \cup \bigcup_{t' \in s_t} \delta^*_R(t'))$. The references effectively form parse trees for all of the strings in $\mathcal{L}(\mathcal{A})$—"all" because of the syntactic correctness requirement, i.e., $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(G)$.

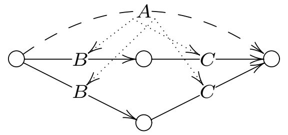
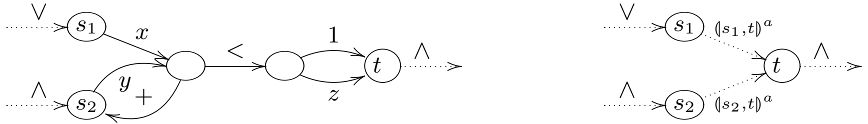


Fig. 4. CFL-reachability



(a) (b)

Fig. 5. Examples for arithmetic and logical FSAs

Let "σ_{ij}" abbreviate "(q_i, σ, q_j)." Because bE cannot be derived from aE in R_G, if $\mathsf{aE}_{ij} \in \delta_N$, then for any string s accepted on a q_i–q_j path over the transitions in $\delta^*_R(\mathsf{aE}_{ij})$, $s \in \mathcal{L}(N, \Sigma, P_G, \mathsf{aE})$. Similarly, if $\mathsf{bE}_{ij} \in \delta_N$, then for any string s generated on a q_i–q_j path, $s \in \mathcal{L}(N, \Sigma, P_G, \mathsf{bE})$. This leads to the following lemma:

Lemma 1. *Each cycle in $\mathcal{A}$, an input to* $\text{V}\small{\text{ALID}}_{FSA,LA,\exists}$, *is either arithmetic (i.e., within $\delta^*_R(\mathsf{aE}_{ij})$ for some $\mathsf{aE}_{ij} \in \delta_N$) or logical (i.e., within $\delta^*_R(\mathsf{bE}_{ij})$ for some $\mathsf{bE}_{ij} \in \delta_N$ and not within $\delta^*_R(\mathsf{aE}_{kl})$ for any $\mathsf{aE}_{kl} \in \delta_N$).*

The subsequent sections present one technique for handling arithmetic cycles and another for logical cycles, but neither technique works for both kinds of cycles. This motivates the primary CFL-reachability-based abstraction used in our algorithm.

Definition 1 (Arithmetic FSA). *Let $\mathcal{A} = (Q, \Sigma, \delta, \delta_R, q_0, q_F)$ and $\mathsf{pred}_{st} \in \delta$. The* arithmetic FSA $(\!|q_s, q_t|\!)^a$ *or $\mathcal{A}_{st} = (Q', \Sigma, \delta', \delta'_R, q_s, q_t)$ where $Q' = \{q \in Q \mid (q, \sigma, q') \in \delta'\} \cup \{q_t\}$, $\delta' = \{t \in \delta \mid t \in \delta'^*_R(\mathsf{pred}_{st})\}$, and $\delta'_R(t) = \delta_R(t)$ if $t \in \delta'$ and $\emptyset$ otherwise.*

Definition 2 (Logical FSA). *Let $\mathcal{A} = (Q, \Sigma, \delta, \delta_R, q_0, q_F)$. The* logical FSA $\mathcal{A}_l =$ *$(Q', \Sigma \cup \mathcal{B}, \delta', \delta'_R, q_0, q_F)$, where $Q' = \{q \in Q \mid (q, \sigma, q') \in \delta'\} \cup \{q_F\}$, $\mathcal{B} = \{(\!|q_i, q_j|\!)^a \mid q_i, q_j \in Q'\}$, $\delta' = \{\sigma_{ij} \in \delta \mid \sigma \neq \mathsf{pred} \wedge \sigma_{ij} \in \delta'^*_R(\mathsf{bE}_{0F})\} \cup \{(\!|q_i, q_j|\!)_{ij} \mid \mathsf{pred}_{ij} \in \delta\}$, $\delta'_R(t \notin \delta') = \emptyset$, and $\delta'_R(t \in \delta') = \bigcup_{s_t \in \delta_R(t)}(\{\{(\!|q_i, q_j|\!)^a_{ij}\}\}$ if $s_t = \{\mathsf{pred}_{ij}\}$; $\{s_t\}$ otherwise).*

The FSA fragment in Fig. 5a has two arithmetic FSAs. The one defined by $(\!|s_1, t|\!)^a$ includes all states and solid transitions in the figure. The one defined by $(\!|s_2, t|\!)^a$ excludes the state s_1 and the x-transition. Figure 5b shows the logical FSA that results from abstracting out the arithmetic FSAs in Fig. 5a. The labels on the states show the correspondence between the original FSA and the logical FSA. Arithmetic FSAs include no logical cycles and logical FSAs include no arithmetic cycles.

We split the problem of validity for FSAs into two subproblems, and in order to define the sub-problems precisely, we define *linear FSAs*:

Definition 3 (Linear FSA). *An FSA $\mathcal{A}$ is a* linear FSA *iff $\mathcal{A}$ is deterministic, $|\mathcal{L}(\mathcal{A})| = 1$, and $\mathcal{A}$ is minimal (i.e., it includes no useless states or transitions).*

The first subproblem takes as input a linear logical FSA and produces a linear arithmetic constraint that is valid if the linear logical FSA accepts a tautology. To this end, Sect. 3 casts arithmetic FSAs as network flow problems. The second subproblem takes as input a logical FSA and produces a finite number of linear logical FSAs such that at least one accepts a tautology iff the input FSA accepts a tautology. Section 4 uses a finite model theorem to unroll logical loops based on the number of variables in the arithmetic FSAs.

3 Arithmetic Loops

We address arithmetic loops by casting questions about arithmetic automata as questions about network flows. The algorithm has four main steps. First, given an arithmetic FSA $\mathcal{A} = (Q, \Sigma, \delta = (\delta_T \cup \delta_N), \delta_R, q_s, q_t)$ we define a labelling function

$$L : \Big(\delta \ \cup \bigcup_{\substack{t \in \delta \\ s_t \in \delta_R(t)}} (t, s_t)\Big) \to \mathcal{F}$$

where $\mathcal{F}$ is a set of *flow variables*. In the constraint that this construction generates, the value of $L(t \in \delta_T)$ equals the number of times the transition t was taken in some accepting path. The value of $L(t, s_t)$ equals the number of times the corresponding derivation occurs in the parse tree of the generated string. The first part of the constraint existentially quantifies the flow variables because the $\text{VALID}_{\text{FSA,LA},\exists}$ problem asks whether *there exist* any tautologies: "$\exists_{f \in \text{codom}(L)} f$."

The second step constrains the values of flow variables so that the values they can take correspond to derivations and paths through $\mathcal{A}$.

$$(1) \bigwedge_{f \in \text{codom}(L)} f \geq 0 \quad \wedge \quad (2) \bigwedge_{t \in \delta_N} L(t) = \sum_{s_t \in \delta_R(t)} L(t, s_t) \quad \wedge$$

$$(3) \bigwedge_{t \in \delta} L(t) = k + \sum_{\substack{t' \in \delta \\ s_{t'} \in \delta_R(t') \\ t \in s_{t'}}} L(t', s_{t'}) \ , \ k = \begin{array}{l} 1 \text{ if } t = \mathsf{pred}_{st} \\ 0 \text{ otherwise} \end{array}$$

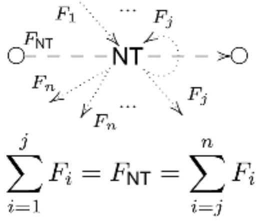


Fig. 6. Flow balancing

Conjunction (1) prohibits solutions that would have a transition being traversed a negative number of times. Figure 6 illustrates how (2) and (3) balance the flow of incoming and outgoing reference transitions.

The third step universally quantifies the variables in $V \cap \Sigma$ because the $\text{VALID}_{\text{FSA,LA},\exists}$ problem asks whether there exists a *tautology*: "$\forall_{v \in V \cap \Sigma}\, v$."

The fourth step uses $\mathsf{C}(\mathsf{pred}_{st})$ to generate a *flow-comparison* constraint that relates the number of times each transition is taken with the value of the generated expression. Because addition commutes, C uses the number of times each term occurs to calculate the value of arithmetic expressions.

$$\mathrm{C}(\mathsf{pred}_{st}) = \bigwedge_{\substack{\{\mathsf{aE}_{si},\mathsf{cmp}_{ij},\mathsf{aE}_{jt}\}\in\delta_R(\mathsf{pred}_{st})\\ \{c_{ij}\}\in\delta_R(\mathsf{cmp}_{ij})}} (L(c_{ij}) = 1) \Rightarrow \mathrm{C}(\mathsf{aE}_{si})\; c\; \mathrm{C}(\mathsf{aE}_{jt}) \qquad \mathrm{C}(\mathsf{aE}_{ij}) = \sum_{\mathsf{aE}_{ij} \rightsquigarrow\!\!\!\twoheadrightarrow^{*} \mathsf{aT}_{kl}} \mathrm{C}(\mathsf{aT}_{kl})$$

$$\mathrm{C}(\mathsf{aT}_{ij}) = \sum_{\{v_{ij}\}\in\delta_R(\mathsf{aT}_{kl})} (L(\mathsf{aT}_{ij}, \{v_{ij}\}) \times v) \quad - \sum_{\{-_{ik},v_{kj}\}\in\delta_R(\mathsf{aT}_{kl})} (L(\mathsf{aT}_{ij}, \{-_{ik}, v_{kj}\}) \times v)$$

Tarski's theorem [4] establishing the decidability of first-order arithmetic guarantees that expressions of this form are decidable when the variables range over real numbers. We state here a completeness result:

Theorem 1. *If the flow-comparison expression generated from an arithmetic FSA* $(\!| s, t |\!)$ *is not valid, then* $(\!| s, t |\!)$ *does not accept a tautology.*

Furthermore, when two or more arithmetic FSAs are linked sequentially by logical operators (e.g., '$\wedge$' or '$\vee$'), we can merge in a natural way the constraints that model the arithmetic FSAs, and the completeness result holds for the sequence of automata.

Theorem 2. *If the flow-comparison expression generated from a linear logical FSA* $\mathcal{A}$ *is not valid, then* $\mathcal{A}$ *does not accept a tautology.*

Unsoundness. If the flow variables ranged over integers, this construction would be sound. Because they range over real numbers, they may take on non-integral values and not correspond to any path through the FSA.

4 Logical Loops

Consider an arithmetic FSA with an $\vee$-transition from its last state to its first state. The arithmetic FSA might not accept any tautology, but two or more passes through the arithmetic FSA joined by '$\vee$' may be a tautology, as in the case of the FSA in Fig. 2a.

4.1 Setup and Loop Unrolling

Unfortunately, we cannot use equations to address logical loops as we did for arithmetic loops. If we did, the generated constraint would not be in first-order arithmetic. Instead, we "unroll" the loop a bounded number of times so that if the loop accepts some tautology, the unrolling must also accept some tautology.

The algorithm for generating linear logical FSAs from a given logical FSA has three main steps. (1) Remove the $\neg$-transitions. (2) Collapse all strongly connected components (SCCs) in the FSA to form a dag, and enumerate the paths through the dag. (3) Transform each collapsed SCC in an FSA $\mathcal{A}_l$ into a linear FSA that replaces the SCC in $\mathcal{A}_l$.

The first step uses graph transformations and an adaptation of DeMorgan's law to propagate '$\neg$' inward. Due to space constraints, the details are omitted here but can be found in the companion technical report [6]. The second step is straightforward, so we omit the details. Section 4.2 presents the third step in detail. Each step preserves the property of accepting a tautology.

The third step takes as input a logical FSA without $\neg$-transitions that only produces syntactically correct strings (as stated in Sect. 2.2). This step relies on the syntactic correctness property, which implies that parentheses are balanced on all paths, and the parenthetic nesting depth is bounded. Because parentheses are always balanced, we can abstract all paths between a pair of matching parentheses into a *parenthetic FSA* $\llparenthesis q_i, q_j \rrparenthesis^p$, which is defined as follows:

Definition 4 (Parenthetic FSA). *Let logical FSA* $\mathcal{A} = (Q, \Sigma, \delta, \delta_R, q_0, q_F)$ *where* $\mathsf{bS}_{st} \in \delta$ *and* $\mathsf{bS}_{st} \rightsquigarrow^{*} \mathsf{bE}_{kl}$. *The* parenthetic FSA $\llparenthesis q_s, q_t \rrparenthesis^p$ *or* $\mathcal{A}_{st} = (Q', \Sigma, \delta', \delta'_R, q_s, q_t)$ *where*

$$\begin{aligned}
Q' &= \{q \in Q \mid (q, \sigma, q') \in \delta'\} \cup \{q_t\} \\
\mathcal{B}_p &= \{\llparenthesis q_i, q_j \rrparenthesis^p \mid q_i, q_j \in Q'\} \\
\delta' &= \{t \in \delta \mid \neg\exists \mathsf{bS}_{ij}.\, \mathsf{bS}_{st} \rightsquigarrow^{+} \mathsf{bS}_{ij} \rightsquigarrow^{+} t\} \\
&\quad \cup \{\llparenthesis q_i, q_j \rrparenthesis^a_{ij} \in \delta \mid \mathsf{bS}_{ij} \in \delta'\} \\
&\quad \cup \{\llparenthesis q_i, q_j \rrparenthesis^p_{ij} \mid \mathsf{bS}_{ij} \in \delta' \wedge \mathsf{bS}_{ij} \rightsquigarrow^{*} \mathsf{bE}_{kl}\} \\
\delta'_R(t \notin \delta') &= \emptyset \\
\delta'_R(t \in \delta') &= \begin{cases} \{\{\llparenthesis q_i, q_j \rrparenthesis^a_{ij}, \llparenthesis q_i, q_j \rrparenthesis^p_{ij}\} \cap \delta'\} & \textit{if}\, \mathsf{bS}_{ij} = t \neq \mathsf{bS}_{st} \\ \delta_R(t) & \textit{otherwise.} \end{cases}
\end{aligned}$$

This abstraction is analogous to the abstraction that defines arithmetic and logical FSAs, except that parenthetic FSAs can be nested within parenthetic FSAs.

After abstracting away parenthetic FSAs, the SCC only has $\vee$- and $\wedge$-transitions and transitions over arithmetic and parenthetics FSAs as atomic units. The following theorem provides the basis for the bounded loop unrolling in Sect. 4.2:

Theorem 3 (Loop Unrolling Theorem). *Let* $T = \{t_1, \cdots, t_m\}$, *where each* t_i *is a comparison of linear arithmetic expressions, and let* n *be the number of distinct variables in* T. *Then* $(\bigvee_{t \in T} t)$ *is a tautology iff there exists some* $T' \subseteq T$ *such that* $|T'| \leq (n + 2)$ *and* $(\bigvee_{t \in T'} t)$ *is a tautology.*

Proof. Helly's theorem states that if $K_1, \cdots, K_m$ are convex sets in n-dimensional Euclidean space $\Re^n$ in which $m \geq n$, and if for every choice of $n + 1$ of the sets K_i there exists a point that belongs to all the chosen sets, then there exists a point that belongs to all the sets $K_1, \cdots, K_m$ [7]. This implies that if $K_1, \cdots, K_m$ are convex sets as before but have no points in common, then there exists some choice of $n + 1$ of the sets K_i that have no points in common.

If $t_1 \vee \cdots \vee t_m$ is a tautology, then by DeMorgan's law, $\neg t_1 \wedge \cdots \wedge \neg t_m$ is unsatisfiable. Each $\neg t_i$ can be rewritten as s_i by replacing the comparison operator with its opposite (e.g., $< \rightleftarrows \geq$). Linear inequalities and linear equalities each define convex spaces (half-spaces and hyperplanes, respectively) in $\Re^n$, where n is the number of variables occurring in them. If each s_i falls into one of these categories (i.e., its comparison operator is one of $\{<, >, \geq, \leq, =\}$), then some choice of $n+1$ of them is unsatisfiable, and the disjunction of the corresponding t_i's is a tautology.

However, a linear disequality (i.e., $\boldsymbol{a} \cdot \boldsymbol{x} \neq b$) defines a non-convex region. Specifically, the points that do not satisfy a disequality lie in a single hyperplane. Suppose that

$$\begin{aligned}
\mathbf{Paren}(\{s_1,\ldots,s_n\}) &= \mathbf{Conj}(s_1) \vee \cdots \vee \mathbf{Conj}(s_n)\\
\mathbf{Conj}(\{b_1,\ldots,b_n\}) &= \boldsymbol{(}\mathbf{Disj}(b_1)\boldsymbol{)} \wedge \cdots \wedge \boldsymbol{(}\mathbf{Disj}(b_n)\boldsymbol{)}\\
\mathbf{Disj}(\llparenthesis q_i, q_j \rrparenthesis^p_{ij}) &= \mathbf{Paren}(\textstyle\bigcup_{t\in\{\mathsf{bT}_{kl}\in\delta_N \mid \mathsf{bE}_{kl}\in\delta^*_R(\mathsf{bS}_{ij})\}} \mathbf{Paths}(t,\emptyset))\\
\mathbf{Disj}(\llparenthesis q_i, q_j \rrparenthesis^a_{ij}) &= \underbrace{\llparenthesis q_i, q_j \rrparenthesis^a \vee \cdots \vee \llparenthesis q_i, q_j \rrparenthesis^a}_{NumVars+2}\\
\mathbf{Paths}(\ t\in\delta_N,\ d\ ) &= \textstyle\bigcup_{s_t\in\delta_R(t)} \mathbf{Zip}(s_t \setminus (d\cup\{t\}), d\cup\{t\})\\
\mathbf{Paths}(\llparenthesis q_i, q_j \rrparenthesis^a_{ij}, d) &= \{\{\llparenthesis q_i, q_j \rrparenthesis^a_{ij}\}\\
\mathbf{Paths}(\llparenthesis q_i, q_j \rrparenthesis^p_{ij}, d) &= \{\{\llparenthesis q_i, q_j \rrparenthesis^p_{ij}\}\}\\
\mathbf{Paths}(\ \wedge_{ij}\ ,\ d\ ) &= \emptyset\\
\mathbf{Zip}(\{t_1,\ldots,t_n\}, d) &= \bigcup_{\substack{s_1\in\mathbf{Paths}(t_1,d)\\ \vdots \\ s_n\in\mathbf{Paths}(t_n,d)}} \{\textstyle\bigcup_{i=1}^n s_i\}
\end{aligned}$$

Fig. 7. Algorithm to construct a linear FSA from an SCC of a logical FSA

for all choices of $n+1$ convex regions (as defined by the s_i's) there exists some point p that satisfies the s_i's. Suppose further that for some choice of $n+1$ convex regions all points common to the region lie in the hyperplane that does not satisfy some disequality s_i. Then a choice of $n+2$ of the s_i's are required for unsatisfiability, and consequently $n+2$ of the t_i's are required for validity.

The only non-convex shape definable by linear constraints has a hyperplane as its region of unsatisfiability. Consequently, given a set S of convex regions whose intersection (is necessarily convex and) is not confined to a hyperplane, no addition of a finite number of non-convex linear constraints to S can cause S to become unsatisfiable. Therefore, no more than $(n+2)$ t_i's will be needed for a tautology. □

4.2 Linearizing Strongly Connected Components

Figure 7 defines five functions: **Paren**, **Conj**, **Disj**, **Paths**, and **Zip**. These five functions are used to construct a linear logical FSA $\mathcal{A}_l$ from a strongly connected component of a logical FSA $\mathcal{A}_s$ such that $\mathcal{A}_l$ accepts a tautology iff $\mathcal{A}_s$ accepts a tautology.

The algorithm to construct $\mathcal{A}_l$ begins as follows. Let $\mathcal{A}_s$ be an SCC without $\neg$-transitions, with parenthetic FSAs abstracted, and with start and final states q_0 and q_F, respectively. Construct a single parenthetic FSA by adding to δ $(q_\alpha, \boldsymbol{(}, q_0)$ and $(q_F, \boldsymbol{)}, q_\beta)$ and letting q_α and q_β be the start and final states, respectively. Begin constructing a linear FSA by calling $\mathbf{Disj}(\llparenthesis q_\alpha, q_\beta \rrparenthesis^p_{\alpha\beta})$. $\mathbf{Disj}$ interprets $(\llparenthesis q_\alpha, q_\beta \rrparenthesis^p_{\alpha\beta})$ as the parenthetic FSA that it represents. The set $\{\mathsf{bT}_{kl}\cdots\}$ over which $\mathbf{Disj}()$ takes a union includes all of the nonterminal transitions from which only conjunctive expressions (i.e., "$a \wedge \cdots \wedge b$") can be derived, but all expressions the can be derived can be entered and exited through $\vee$-transitions. The **Paths** function then returns a set S of sets of transitions, where each set s of transitions includes all of the arithmetic and parenthetic FSAs on some shortest (i.e., $\neg\exists s' \in S. s' \subset s$) acyclic path derived from the transition t. Because $\mathcal{A}_s$ is strongly connected, each path represented by the set returned

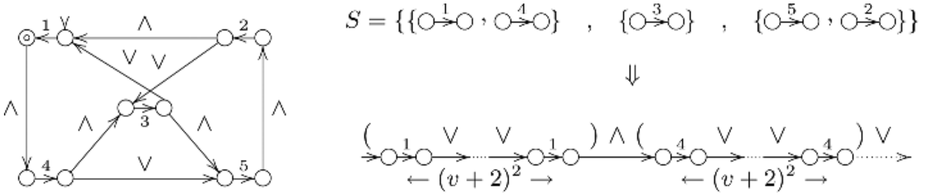


Fig. 8. An example SCC and the result of Fig. 7, where $v = \mathit{NumVars}$

from Paths can be entered and exited through '∨,' and disjunction weakens expressions monotonically, if a tautology can be constructed from the conjunctive expressions returned from Paths, then $\mathcal{A}_s$ accepts a tautology. Because conjunction strengthens expressions monotonically, and the conjunctive expressions returned from Paths are all Paths returns all shortest conjunctive expressions, if $\mathcal{A}_s$ accepts some tautology, then a tautology can be constructed by the shortest conjunctive expressions.

Paths passes the set representing all shortest conjunctive expressions to Paren, which begins to construct a linear structure by calling Conj on each expression, and connecting the results with '∨.' A DNF expression constructed by disjoining several instances of one of these conjunctive expressions can be refactored into a CNF expression. Conj constructs such a CNF expression. Because it constructs a CNF expression, each element (arithmetic or parenthetic FSA) in the set can be handled individually and independently by Disj. If the element is a parenthetic FSA, Disj recurses down and produces a linear construction based on the parenthetic FSA. If the element is an arithmetic FSA, Disj disjoins $\mathit{NumVars} + 2$ copies of it, where $\mathit{NumVars}$ is the number of distinct variables that appear in the original FSA (i.e., $|\{v \in V \mid v_{ij} \in \delta\}|$). Theorem 3 implies that if any (necessarily finite) disjunction of constraints from a given set constitutes a tautology, then at most $\mathit{NumVars} + 2$ of the constraints are needed to construct a tautology. Given a complete linear structure, a linear logical FSA can be constructed by using the sequence of tokens as the labels for the transitions in a linear FSA.

To illustrate the algorithm, Fig. 8 shows an example SCC, where numbers (1–5) represent arithmetic FSAs. The set S consists of three sets, and below that, the beginning of the constructed linear FSA shows how those sets are used. Note that each set in S consists of the arithmetic FSAs along a path that can be entered and exited from ∨-transitions but has only ∧-transitions between arithmetic FSAs.

4.3 Soundness, Completeness, and Complexity

Taken together, the algorithms for constructing linear logical FSAs from a logical FSA are sound and complete for finding tautologies in logical FSAs.

Theorem 4 (Soundness and Completeness). *Given an FSA $\mathcal{A}$ where $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(G)$, the algorithm presented in Sect. 4 produces a finite set $S_\mathcal{A}$ of linear logical FSAs such that there exists an FSA in $S_\mathcal{A}$ that accepts a tautology iff $\mathcal{A}$ accepts a tautology.*

Proof. The abstraction from $\mathcal{A}$ to a logical FSA $\mathcal{A}'$ described in Sect. 2.2 maintains language equivalence if a path through $\mathcal{A}'$ includes paths through the arithmetic FSAs that correspond to their names. The algorithm to remove ¬-transitions produces a logical FSA $\mathcal{A}^+$ from $\mathcal{A}'$ such that there exists a bijective mapping $b : \mathcal{L}(\mathcal{A}^+) \to \mathcal{L}(\mathcal{A}')$

where $b(\varphi^{+}) = \varphi'$ implies $\varphi^{+} \equiv \varphi'$. Collapsing SCCs and finding paths through the dag produces a finite set $S_{\mathcal{A}^{+}}$ of FSAs from $\mathcal{A}^{+}$ such that a path in $\mathcal{A}^{+}$ can be mapped to a path in some $\mathcal{A}_S \in S_{\mathcal{A}^{+}}$, and vice versa. The algorithm in Fig. 7 then produces a finite set $S_{\mathcal{A}}$ of linear logical FSAs from $S_{\mathcal{A}^{+}}$, such that, by the algorithm in Sect. 4.2 and Theorem 3, there exists an FSA $\mathcal{A}^{-} \in S_{\mathcal{A}}$ that accepts a tautology iff some $\mathcal{A}_S \in S_{\mathcal{A}^{+}}$ accepts a tautology. □

A logical FSA is no larger than the FSA from which it is abstracted. Removing $\neg$-transitions produces at most a constant number of instances of each state, so the resulting FSA has size $O(|Q|)$ in the size of the input. The states in the FSA can be partitioned into Q_q, those states that can be reached from themselves, Q_p, those that cannot. Let $q = |Q_q|$ and $p = |Q_p|$. Collapsing SCCs and enumerating all paths generates $O(2^p)$ paths. From each of these paths, the algorithm in Fig. 7 produces linear logical FSAs. If $p \gg q$, then each path has length $O(p)$. Otherwise, each path is bounded by $|S|$, which is $O(2^q)$, and the length of the FSA produced from each $S \in \int$, which is $O(n(q2^q))$, where n is the number of unique variables in the arithmetic FSAs. So, each path has length $O(\max(p, nq2^q))$. From each path a query, which is linear in the size of the path, is created and sent to a first-order arithmetic decision procedure.

5 Related Work

This section surveys closely related work.

First-Order Theories. Tarski established the decidability of the first-order theory of real numbers with addition and multiplication through quantifier elimination [4]. Collins used cylindrical decomposition to check validity in the same theory more efficiently, but his algorithm also has high complexity [8]. The first-order theory over integers is undecidable because of the undecidability of solving Diophantine equations [3]. However, an important fragment, Presburger Arithmetic, is decidable [9].

Linear Constraints. In program analysis and formal verification, decision procedures for linear constraints are widely used. Some proposed techniques include Fourier-Motzkin variable elimination [10], the Sup-Inf method of Bledsoe [11], and Nelson's method based on Simplex [12]. More tractable algorithms can be found by restricting the class of integer constraints further. Pratt gives a polynomial time algorithm for the form of linear constraints $x \leq y + k$, where k is an integer [13]. Shostak considers a slightly more general problem $ax+by \leq k$, where a, b, and k are integer constants [14]. He uses "loop residues" for an algorithm which requires exponential time in the worst case. Aspvall and Shiloach give a refined algorithm for the same form which runs in polynomial time [15]. Su and Wagner [16] leverage ideas from Pratt and Shostak to propose the first polynomial time algorithm for a general class of integer range constraints underlying the standard example (range constraints [17]) of abstract interpretation [18].

Combined Theories. In 1979, Nelson and Oppen proposed a method for combining theories in a decision procedure [19]. Contemporary theorem provers, such that as in Necula and Lee's certifying compiler [20], use Nelson and Oppen's architecture for cooperating decision procedures. In 1984, Shostak introduced an algorithm for deciding the satisfiability of quantifier-free formulas in a combined theory [21]. This algorithm improved over previous decision procedures by enabling multiple theories to be integrated uniformly instead of using separate, communicating processes. This algorithm serves as the basis for decision procedures found in several tools including PVS [22], STeP [23], and SVC [24]. SVC uses a decision procedure for a fragment of first-order logic which excludes quantifiers, but includes equality, uninterpreted functions and constants, arrays, records, and bit-vectors, as well as propositional connectives. CVC Lite [25] is a descendant of SVC that includes a builtin SAT solver and support for quantifiers.

Helly-like Theorems. Helly-like theorems have been used to improve certain individual linear programming problems, such as finding a point in the intersection of a family of convex sets [7,26]. In 1994, Amenta proved a general relation between Helly-like theorems and generalized linear programming [27]. None of these, however, use Helly's theorem as this paper does: to bound the number of constraints needed to find a tautology in unboundedly large sets of constraints.

6 Conclusions and Future Work

We have introduced the class of decision problems for language generators $\textsc{Valid}_{\Pi,\Phi,K}$ (motivated by the need for advanced checking of meta-programs) and an algorithm for $\textsc{Valid}_{\mathrm{FSA},\mathrm{LA},\exists}$. Our algorithm is based on casting FSAs as network flow problems and leveraging a novel application of Helly's theorem to bound the number of comparison expressions needed for a tautology. The network flow-based construction is unsound because the flow variables may take on non-integral values.

This paper opens up several interesting directions for future work. First, language generators that can match calls and returns, such as tree automata and push-down automata, are better suited for certain program analysis problems in meta-programming than finite state automata. Because the algorithm presented here relies on the boundedness of parenthetic nesting, new insights will be needed to construct algorithms over these more expressive formalisms. Second, we expect that similar techniques to the ones presented here will yield an algorithm for $\textsc{Valid}_{\mathrm{FSA},\mathrm{LA},\forall}$. Third, this algorithm does not exploit much of the finer-grained structure of the FSA. We expect that this can be used to provide an alternative, and frequently lower, bound on the number of expressions needed for a tautology. Fourth, we are interested in studying the relation of $\textsc{Valid}_{\mathrm{FSA},\mathrm{LA},\exists}$ to MSO logic, which also has an automata-based formulation. Finally, we would like to find matching upper and lower bounds for the $\textsc{Valid}_{\mathrm{FSA},\mathrm{LA},\exists}$ problem in order to know its exact complexity.

References

1. Borland, M.: Advanced SQL Command Injection: Applying defense-in-depth practices in web-enabled database applications (2002)
2. Christensen, A.S., Møller, A., Schwartzbach, M.I.: Precise analysis of string expressions. In: Proc. SAS'03. (2003) 1–18 URL: `http://www.brics.dk/JSA/`.

3. Matiyasevich, Y.: Solution of the tenth problem of Hilbert. Mat. Lapok **21** (1970) 83–87
4. Tarski, A.: A Decision Method for Elementary Algebra and Geometry. University of California Press (1951)
5. Gould, C., Su, Z., Devanbu, P.: Static checking of dynamically generated queries in database applications. In: Proc. ICSE'04. (2004)
6. Wassermann, G., Su, Z.: Validity Checking for Finite Automata over Linear Arithmetic. Technical report, University of California, Davis, Computer Science Dept. (2006)
7. Danzer, L., Grünbaum, B., Klee, V.: Helly's theorem and its relatives. In: Proceedings of the Symposium on Pure Mathematics. Volume 7 of Convexity., AMS (1963) 101–180
8. Collins, G.E.: Hauptvortrag: Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In: A. Theory and Formal Languages. (1975)
9. Wolper, P., Boigelot, B.: An automata-theoretic approach to Presburger arithmetic constraints (extended abstract). In: SAS, Springer-Verlag (1995) 21–32
10. Pugh, W.: The omega test: a fast and practical integer programming algorithm for dependence analysis. In: Proc. Supercomputing. (1991) 4–13
11. Bledsoe, W.: The Sup-Inf method in Presburger arithmetic. Technical report, University of Texas Math Department (1974)
12. Nelson, G.: Techniques for program verification. Technical report, Xerox PARC (1981)
13. Pratt, V.: Two easy theories whose combination is hard. Technical report, MIT (1977)
14. Shostak, R.: Deciding linear inequalities by computing loop residues. J. ACM **28** (1981)
15. Aspvall, B., Shiloach, Y.: A polynomial time algorithm for solving systems of linear inequalities with two variables per inequality. SIAM Computing **9** (1980) 827–845
16. Su, Z., Wagner, D.: A class of polynomially solvable range constraints for interval analysis without widenings and narrowings. In: Proc. TACAS'04. (2004)
17. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Symposium on Programming. (1976) 106–130
18. Cousot, P., Cousot, R.: Abstract interpretation: A unified lattice model for static analysis of programs by construction or approximation of fixpoints. In: POPL. (1977) 234–252
19. Nelson, G., Oppen, D.C.: Simplification by cooperating decision procedures. TOPLAS **1** (1979) 245–257
20. Necula, G.C., Lee, P.: The design and implementation of a certifying compiler. In: Proc. PLDI. (1998)
21. Shostak, R.E.: Deciding combinations of theories. J. ACM **31** (1984) 1–12
22. Owre, S., Shankar, N., Rushby, J.: PVS: A Prototype Verification System. In: Proc. CADE 11. (1992)
23. Bjørner, N., Browne, A., Chang, E., Colón, M., Kapur, A., Manna, Z., Sipma, H., Uribe, T.E.: STeP: Deductive-algorithmic verification of reactive and real-time systems. In: Proc. CAV. (1996)
24. Barrett, C.W., Dill, D.L., Levitt, J.R.: Validity Checking for Combinations of Theories with Equality. In: Proc. FMCAD. (1996) 187–201
25. Barrett, C., Berezin, S.: CVC Lite: A new implementation of the cooperating validity checker. In: Proc. CAV. (2004)
26. Avis, D., Houle, M.E.: Computational aspects of Helly's theorem and its relatives. International Journal of Computational Geometry Applications **5** (1995) 357–367
27. Amenta, N.: Helly-type theorems and generalized linear programming. Discrete & Computational Geometry **12** (1994) 241–261

Game Semantics for Higher-Order Concurrency

J. Laird*

Department of Informatics, University of Sussex, UK

Abstract. We describe a denotational (game) semantics for a call-by-value functional language with multiple threads of control, which may communicate values of general type on locally declared channels.

This develops previous work which interpreted freshly generated names in a category of games acted upon by the group of natural number automorphisms, by showing how names may be associated with "dependent arenas" in which interaction between strategies, corresponding to asynchronous communication on named channels, may occur.

We describe a model of the call-by-value λ-calculus (a closed Freyd category) based on these arenas, and use this as the basis for interpreting our language. We prove that the semantics is fully abstract with respect to may-testing using a correspondence between channel and function types based on the "triggering" representation of procedure-passing in terms of name-passing.

1 Introduction

Higher-order concurrency — the capacity to generate multiple threads of control and pass higher-order functions and processes as values between them — is a powerful and subtle programming paradigm. Languages and calculi with these features, such as *Concurrent ML* and the higher-order π-calculus, have been extensively studied using operational methods, but the combination of dynamic name creation and higher-order value-passing has presented a long standing challenge for denotational semantics. In this paper, we shall develop a denotational model of a call-by-value functional language with higher-order concurrency, including dynamically generated channel names, and prove that it is fully abstract with respect to may-testing.

Our model is based on *game semantics*. This has proved successful in giving precise (fully abstract) models of higher-order programming languages with many different features, including concurrency [3,12]. Our semantics opens the door to a range of (largely theoretical) applications: by formalizing some of the categorical and algebraic structures required to capture higher-order concurrency, it can contribute to the development of general theories, whilst also potentially being the basis for more specific forms of program analysis, already developed for various games models, such as control and information flow analysis [14], and model-checking of properties such as program equivalence [2].

* E-mail: jiml@sussex.ac.uk. Research supported by UK EPSRC grant GR/S72181.

S. Arun-Kumar and N. Garg (Eds.): FSTTCS 2006, LNCS 4337, pp. 417–428, 2006.

1.1 Related Work

Our semantics is given for a language with syntax based on Reppy's Concurrent ML [17]: it may also be viewed as a programming language variant of the higher-order π-calculus [18]. Both languages (or fragments thereof) have been investigated using operational techniques [18,1,8,9], giving, for example, labelled transition systems which are closely related to game semantics. Indeed, our semantics may be viewed as a trace semantics, defined compositionally. (See [12] for an explicit comparison between game and trace semantics of the π-calculus.)

Our model uses (and adapts) a variety of notions from game semantics, including the representation of call-by-value types introduced by Honda and Yoshida [5]. In common with earlier models of shared-variable concurrency [3] and the π-calculus [12], we represent terms up to asynchronous, may-testing observation as sets of justified sequences (traces) closed under a preorder. Particularly significant for the current work is the development of a category of "ν-arenas and ν-strategies" acted upon by the group of natural number automorphisms [11], with which the manipulation and generation of *names* can be interpreted (in the current case, channel names). A certain degree of parametricity is implicit in the representation of values at a range of types as names, and there are parallels in this respect between our games and the game semantics of polymorphism described by Hughes [6].

1.2 Contribution of This Paper

The main technical contribution of this paper is to show how the game semantics of freshly generated names developed in [11] may be used to model the passing of values of all types on named, typed channels. The construction which makes this possible is a notion of "tree arena" which has ν-arenas as its nodes, each of which has as its children a "dependent arena" for each name which may be mentioned by each move. Using a name allows interaction to take place in its dependent arena, corresponding to message passing on the associated channel. We define a category of games in which we define "parallel composition plus hiding" of strategies to allow interaction both at top level, and within dependent arenas with names which have been made public. We then define the structure of a categorical model of the call-by-value λ-calculus (a premonoidal closed category), and interpret the key operations of our language (spawning of threads, generation of channels, sending and receiving of messages) as simple strategies. We show that our interpretation is sound and adequate with respect to may-testing.

We then prove that the bounded elements of our semantics are definable as terms of $\mathcal{L}$, and hence that it is fully abstract with respect to may-testing. The key to the proof of definability is the observation that justification pointers may be encoded as names using a series of definable retractions. This is a semantic counterpart of Sangiorgi's *triggering* translation from higher-order processes into the π-calculus [18]. We may then give a simple proof that "pointer-free" sequences are definable as π-calculus-like terms.

2 A Language with Higher-Order Concurrency

The programming language $\mathcal{L}$ which we shall interpret contains several of the key features of CML [17]: new thread generation, new channel declaration (it omits thread identifiers, which are readily expressible using channel names, and event types). Thus it is similar both to μCML [1] and $\mu\nu$CML [9] (although, unlike these languages, communication is *asynchronous*). The types of $\mathcal{L}$ are given by the grammar:

$S, T := B \mid S * T \mid S \Rightarrow T \mid \mathtt{chan}[T]$ where B is a set of basic types including at least $\mathtt{unit}$, $\mathtt{bool}$, and an empty type $\mathtt{0}$. The syntax and typing judgements of $\mathcal{L}$ are those of the typed λ-calculus extended with the following constants:

Pairing/Projection $\mathtt{pair} : S \Rightarrow T \Rightarrow S * T$, $\mathtt{fst} : S * T \Rightarrow S$, $\mathtt{snd} : S * T \Rightarrow T$
Atomic Values $() : \mathtt{unit}$ and $\mathtt{tt}, \mathtt{ff} : \mathtt{bool}$,
Conditional $\mathtt{If} : \mathtt{bool} \Rightarrow (T * T) \Rightarrow T$
Equality Testing $\mathtt{eq} : \mathtt{chan}[T] * \mathtt{chan}[T] \Rightarrow \mathtt{bool}$,
Channel Declaration $\mathtt{new}_T : \mathtt{unit} \Rightarrow \mathtt{chan}[T]$,
Thread Creation $\mathtt{spawn} : (\mathtt{unit} \Rightarrow \mathtt{0}) \Rightarrow \mathtt{unit}$,
Message Passing $\mathtt{send} : \mathtt{chan}[T] * T \Rightarrow \mathtt{0}$ and $\mathtt{recv} : \mathtt{chan}[T] \Rightarrow T$

We write (M, N) for $(\mathtt{pair}\, M)\, N$, $\nu x.M$ for $(\lambda x.M)\, (\mathtt{new}\, ())$, $\mathtt{nil}$ for $\nu c.\mathtt{recv}\, c$, $M = N$ for $(\mathtt{eq}\, (M, N)$ and $M|N$ for $(\mathtt{spawn}\, (\lambda x.M)); N$. We may express recursively defined functions as: $\mu f.\lambda x.M =_{df}$
$\nu c.\mathtt{let}\, f = (\lambda x.\mathtt{let}\, g = \mathtt{recv}\, c\ \mathtt{in}\ (\mathtt{send}\, (c, g)|g\, x))\ \mathtt{in}\ (\mathtt{send}\, (c, \lambda x.M)|f)$.
In particular, we define the replicated send: $!\mathtt{send}\, M {=_{df}} \mu f.\lambda x.(\mathtt{send}\ (c, M)|f\ ())$.

A *configuration* of consists of a multiset $\mathcal{T}$ of programs or *threads* $M_1, \ldots, M_n$ (of which at most one has non-empty type), and a set $\mathcal{N}$ of typed channel names such that $\mathcal{N} \vdash M_i$ for $1 \leq i \leq n$. The *values* of $\mathcal{L}$ are given by the grammar: $U, V ::= v \mid n \mid C \mid \mathtt{pair}\, U \mid (\mathtt{pair}\, U)\, V \mid \lambda x.M$
(where v ranges over base type values, n over channel names and C over constants). The *evaluation contexts* are: $E[\_] ::= [\_] \mid E[\_]\, M \mid V\, E[\_]$.
The "small step" evaluation rules for evaluating configurations are as follows:

$$\begin{array}{lcl}
\mathcal{T}, E[(\lambda x.M)\, V], \mathcal{N} & \longrightarrow & \mathcal{T}, E[M[V/x]], \mathcal{N} \\
\mathcal{T}, E[\mathtt{fst}(U, V)], \mathcal{N} & \longrightarrow & \mathcal{T}, E[U], \mathcal{N} \\
\mathcal{T}, E[\mathtt{snd}(U, V)], \mathcal{N} & \longrightarrow & \mathcal{T}, E[V], \mathcal{N} \\
\mathcal{T}, E[\mathtt{eq}(a, a)], \mathcal{N} & \longrightarrow & \mathcal{T}, E[\mathtt{tt}], \mathcal{N} \\
\mathcal{T}, E[\mathtt{eq}(a, b)], \mathcal{N} & \longrightarrow & \mathcal{T}, E[\mathtt{ff}], \mathcal{N},\ \text{if } a \neq b \\
\mathcal{T}, E[\mathtt{If}\ \mathtt{tt}], \mathcal{N} & \longrightarrow & \mathcal{T}, E[\mathtt{fst}], \mathcal{N} \\
\mathcal{T}, E[\mathtt{If}\ \mathtt{ff})], \mathcal{N} & \longrightarrow & \mathcal{T}, E[\mathtt{snd}], \mathcal{N} \\
\mathcal{T}, E_1[\mathtt{send}(a, V)], E_2[\mathtt{recv}(a)], \mathcal{N} & \longrightarrow & \mathcal{T}, E_2[V], \mathcal{N} \\
\mathcal{T}, E[\mathtt{spawn}(V)], \mathcal{N} & \longrightarrow & \mathcal{T}, V\, (), E[()], \mathcal{N} \\
\mathcal{T}, E[\mathtt{new}_T\, ()], \mathcal{N} & \longrightarrow & \mathcal{T}, E[a], \mathcal{N} \cup \{(a, \mathtt{chan}[T])\} \quad a \notin \mathcal{N}
\end{array}$$

We write $M \Downarrow$ (M *may* converge) if $M, 0 \twoheadrightarrow (\mathcal{T}, V), \mathcal{N}$ for some $\mathcal{T}, V, \mathcal{N}$. Thus we define observational approximation and equivalence with respect to may-testing: $M \lesssim N$ if $C[M] \Downarrow$ implies $C[N] \Downarrow$ for all compatible closing contexts $C[\_] : \mathtt{unit}$. $M \simeq N$ if $M \lesssim N$ and $N \lesssim M$.

3 Game Semantics

Our notion of game is based on the dialogue games of Hyland and Ong [7], developed in e.g. [15,5] and extended in [11] with structure for manipulating a countable set of names, in the form of an action of the automorphism group of the natural numbers. A significant departure from these games for sequential languages arises because in the concurrent setting there are moves which may be played by either participant in a dialogue. So polarity (Player/Opponent labelling) is not intrinsic to arenas but to interactions.

An (underlying) arena A is a tuple $(M_A, \lambda_A, \vdash_A)$ consisting of a set of moves M_A, a question/answer labelling $\lambda_A : M_A \rightarrow \{Q, A\}$ and an *enabling relation* $\vdash_A \subseteq M_A \times M_A$ such that no answer enables an answer. We write M_A^I for the subset of M_A of consisting of moves with no enabling move (the *initial* moves), and say that an arena is *A-rooted* if all such moves are answers.

A justified sequence over the arena A is a sequence of moves of A together with a "justification pointer" from each non-initial move to some enabling move.

Definition 1. *A (partial or total) polarization for a justifed sequence s is a (partial or total) labelling of the occurrences of moves in s as belonging to Player and Opponent (concretely, a function λ^{OP} from the non-empty prefixes of s to $\{P, O\}$) such that the justifier of any Player move is an Opponent move and vice-versa. A polarized sequence t is a justified sequence with a total polarization. We write $t^{\perp}$ for the polarized sequence in which the labelling is reversed.*

We now recall the notion of ν-arena introduced in [11]. Let G be the topological group of automorphisms on $\mathbb{N}$ with the product topology on $\mathbb{N}^{\mathbb{N}}$. An action of G on a set A is continuous (with respect to the discrete topology on A) iff the stabiliser of any $a \in A$ is open in G and thus equal to the stabiliser of a finite subset $\nu(a) \subseteq \mathbb{N}$, the *support* of a.

Definition 2. *A ν-arena (A, π) is an underlying arena A together with a continuous action of G on M_A $(\cdot)$ such that $\lambda_A(\pi \cdot m) = \lambda_A(m)$ and $m \vdash n$ iff $\pi \cdot m \vdash \pi \cdot n$, which therefore extends pointwise to a continuous action on justified sequences of A. We write $\sim$ for the equivalence relation determined by this action — i.e. $s \sim t$ if $\exists \pi \in G. \pi \cdot s = t$.*

A key example is the ν-arena of names N, in which the set of moves is the set of natural numbers (all of which are initial moves), with the canonical action of G upon $\mathbb{N}$ — i.e. $M_N = M_N^I = \mathbb{N}$, $\lambda(i) = A$ for all i, and $\pi \cdot i = \pi(i)$.

For each polarized sequence s, we define the sets $P_\nu, O_\nu \subseteq_{fin} \mathbb{N}$ of new names introduced by Player and Opponent in s. $P_\nu(\varepsilon) = \varnothing$ and:

- $P_\nu(sa) = P_\nu(s) \cup (\nu(sa) - \nu(s))$ if a is Player move,
- $P_\nu(sa) = P_\nu(s)$ otherwise.
- $O_\nu(s) = \nu(s) - P_\nu(s)$.

In order to use names to represent channel types, we introduce a notion of "tree arena", in which each node is an arena, from which there are branches for each name occurring in the support of each move. Essentially, playing a move which mentions a name with a given "dependent arena" allows both Player and Opponent to commence play in that arena, corresponding to sending (if Player starts) or receiving (if Opponent starts) a value on the associated channel.

Definition 3. *A (finite-depth) tree arena is a ν-arena A, together with an indexed set $\{\alpha(m)_i \mid i \in \nu(m)\}$ of tree arenas for each move $m \in M_A$, such that:*

- *G-Invariance: for any $\pi \in G$, $m \in M_A$ and $i \in \nu(m)$, $\alpha(m)_i = \alpha(\pi \cdot m)_{\pi(i)}$.*
- *Finite Depth: there is no infinite chain of arenas $A \ll A_1 \ll A_2 \ll \ldots$, where $B \ll C$ if there exists $m \in M_B$ and $i \in \nu(m)$ such that $\alpha(m)_i = C$.*

Thus, for example, for any tree arena A, we may form the tree arena $Ch(A)$ which has as its root node the arena N, and as its children, copies of the arena A — i.e. $ch(A) = (N, \{\{\alpha(i)_i\} \mid i \in \mathbb{N}\})$, where $\alpha(i)_i = A$ for all i.

We refer to the set of nodes of a tree arena as its *dependent arenas*. Formally, this is defined by induction on tree depth as follows:
$|A| = \{\alpha(m)_i \mid m \in M_A \wedge i \in \nu(m)\} \cup \bigcup\{|\alpha(m)_i| \mid m \in M_A \wedge i \in \nu(m)\}$.

We obtain a tree arena $\widehat{A}$ — the *expansion of* A — by explicitly adding $\mathbb{N}$-indexed copies of the dependent arenas of A to the top node. More precisely:

- $M_{\widehat{A}} = M_A + \{\langle i, A, m\rangle \in \mathbb{N} \times |A| \times \bigcup\{M_B \mid B \in |A|\} \mid m \in M_B \wedge i \notin \nu(m)\}$,
- $m \vdash_{\widehat{A}} \mathrm{in_l}(n)$ if $m = \mathrm{in_l}(m')$ and $m' \vdash_A n$.
  $m \vdash_{\widehat{A}} \mathrm{in_r}(\langle i, B, n\rangle)$, if $m = \mathrm{in_r}(\langle i, B, m'\rangle)$, where $m' \vdash_B n$,
- $\lambda_{\widehat{A}}(\mathrm{in_l}(m)) = \lambda_A(m)$ and $\lambda_{\widehat{A}}(\mathrm{in_r}(\langle i, B, m\rangle)) = \lambda_B(m)$,
- $\pi \cdot \mathrm{in_l}(m) = \mathrm{in_l}(\pi \cdot_A m)$ and $\pi \cdot \mathrm{in_r}(\langle i, B, m\rangle) = \mathrm{in_r}(\pi(i), \pi \cdot_B m)$
- $\alpha^{\widehat{A}}(\mathrm{in_l}(m))_i = \alpha^A(m)_i$
  $\alpha^{\widehat{A}}(\mathrm{in_r}(\langle i, B, m\rangle))_i = B$, $\alpha^{\widehat{A}}(\mathrm{in_r}(\langle i, B, m\rangle))_j = \alpha^B(m)_j$ if $j \neq i$.

Names of the form $\langle i, m\rangle$ are called *dependent moves.* The name of $\langle i, m\rangle$ is i.

Definition 4. *A* legal sequence *on A is a justified sequence s on $\widehat{A}$, satisfying:*

Uniformity. *Every occurrence of a name refers to the same arena: if $ta, t'a' \sqsubseteq s$ and $i \in \nu(a) \cap \nu(a')$ then $\alpha(a)_i = \alpha(a')_i$.*
Dependency. *The name of any dependent move has already occurred in s: if $t\langle i, a\rangle \sqsubseteq s$ then $i \in \nu(t)$.*
Well-openedness. *s contains at most one initial and non-dependent move.*
Well-answering. *Every question in s justifies at most one answer.*

A negative *sequence is a legal sequence starting with an Opponent move. We write L_A for the legal sequences over A, and L_A^- for the negative sequences.*

To represent the behaviour of strategies "up to asynchronous observation" requires saturation under a preorder $\preceq$ on polarized sequences as in e.g. [10]. This is defined to be the least preorder on polarized sequences such that:

- If $\lambda(a) = O$ and $P_\nu(sat) = P_\nu(st)$ then $sabt \preceq sbat$ and if $\lambda^{OP}(a) = P$ and $O_\nu(sat) = O_\nu(st)$ then $sbat \preceq sabt$.

– If $\lambda(a) = O$ and $P_\nu(sat) = P_\nu(st)$ then $sat \preceq st$, and if $\lambda(a) = P$ and $O_\nu(sat) = O_\nu(st)$, then $t \preceq sat$.

Definition 5. *Let A be a tree arena. A strategy $\sigma : A$ is a non-empty set of negative sequences over A which is prefix closed, $\sim$-closed and $\preceq$-closed. We write* $\ker(\sigma)$ *for the set of $\preceq$-minimal sequences of σ — i.e. $\{s \in \sigma \mid \forall t \in \sigma.s \preceq t \Longrightarrow t \preceq s\}$.*

4 Denotational Semantics

We will now construct a *premonoidal* closed category [16] of tree arenas and strategies in which to model the call-by-value λ-calculus. This follows the constructions of Honda and Yoshida [5] or variants described by in [13], and (for ν-arenas) [11]. In each case the group action and dependent arena structure on compound arenas is defined pointwise. Play in the function-space arena $A_1 \rightarrow A_2$ starts on the left (by labelling the initial moves of A_1 as questions which enable the initial moves of A_2).

Definition 6. *Given tree arenas A_1, A_2 we define the (Q-rooted) call-by-value function-space tree-arena $A_1 \rightarrow A_2$ as follows:*

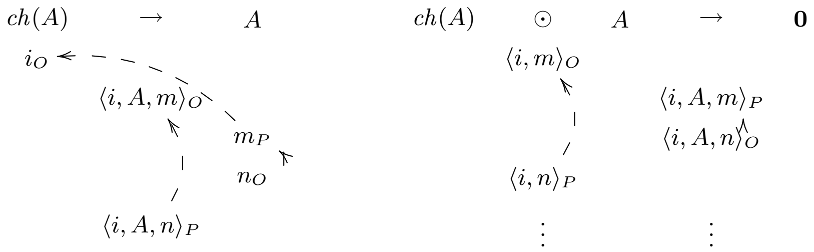
- $M_{A_1 \rightarrow A_2} = M_{A_1} + M_{A_2}$,
- $\lambda_{A_1 \rightarrow A_2}(\mathrm{in}_i(m)) = Q$, *if* $i = 1$ *and* $m \in M^I_{A_2}$,
  $\lambda_{A_1 \rightarrow A_2}(\mathrm{in}_i(m)) = \lambda_{A_i}(m)$, *otherwise,*
- $m \vdash_{A_1 \rightarrow A_2} \mathrm{in}_i(n)$ *if* $m = \mathrm{in}_i(m')$ *and* $m' \vdash_{A_i} n$ *or* $i = 2$, $n \in M^I_{A_2}$ *and* $m = \mathrm{in}_1(m')$, *where* $m' \in M^I_{A_1}$.
- $\pi \cdot \mathrm{in}_i(m) = \mathrm{in}_i(\pi \cdot m)$.
- $\alpha(\mathrm{in}_i(m))_j = \alpha^{A_i}(m)_j$.

For example, for each arena A, we have a “channel creation” strategy $\mathsf{new} : I \rightarrow ch(A)$ (where I is the arena I with a single initial answer move). This responds to Opponent’s initial question by generating a fresh name and making it public (i.e. playing an arbitrary move in N) — thus $\ker(\mathsf{new}) = \{\varepsilon, q\} \cup \{qi \mid i \in \mathbb{N}\}$. For each A-rooted arena A, we have a strategy $\mathsf{recv} : ch(A) \rightarrow A$ which responds to Opponent’s initial question — which supplies the a channel name i — by playing copycat between A and the dependent arena of i. (See Fig. 1.)

To define the composition of strategies $\sigma : A \rightarrow B$ and $\tau : B \rightarrow C$ we need to allow interaction both in the shared “public arena” B and in the dependent arenas. In addition, we impose the “freshness conditions” introduced in [11] to ensure that the new names introduced by σ are disjoint from those introduced by τ, and that both are disjoint from those introduced by Opponent.

Definition 7. *An interaction sequence is a justified sequence t with two partial polarizations λ^L and λ^R such that:*

- *Every move has at least one polarity: for all $s \sqsubseteq t.\lambda^L(s) \downarrow$ or $\lambda^R(s) \downarrow$.*
- *There is no $s \sqsubseteq t$ such that $\lambda^L(s) = \lambda^R(s)$.*
- $P_\nu(s{\restriction}L) \cap P_\nu(s{\restriction}R) = (P_\nu(s{\restriction}L) \cup P_\nu(s{\restriction}R)) \cap O_\nu(s{\restriction}L \triangle R) = \varnothing$,



$ch(A) \to A$
i_O
$\langle i, A, m\rangle_O$
m_P
n_O
$\langle i, A, n\rangle_P$
$ch(A) \odot A \to \mathbf{0}$
$\langle i, m\rangle_O$
$\langle i, n\rangle_P$
$\langle i, A, m\rangle_P$
$\langle i, A, n\rangle_O$

Fig. 1. Typical plays of recv and snd

(Where we write $s{\restriction}L$ for the polarized sequence obtained by restricting to moves for which λ^L is defined, and $s{\restriction}L\triangle R$ for the restriction to moves for which only one of λ^L, λ^R is defined.)

Let $I_{A,B,C}$ be the set of interaction sequences which are legal sequences of $(A \to B) \to C$. Given $\sigma : A \to B, \tau : B \to C$, we may now define:
$\sigma;\tau : A \to C = \{s \in L^-_{A\to C} \mid \exists t \in I_{A,B,C}.t{\restriction}L \in \sigma \wedge t{\restriction}R \in \tau \wedge s = t{\restriction}L\triangle R\}$.

We prove that composition is well-defined and associative following the standard arguments used in [7,15,5], extended to ν-strategies in [11]. The identity strategy $\mathsf{id}_A : A \to A$ is defined: $\ker(\mathsf{id}_A) = \{s \in (L^-_{A\to A})^E \mid \forall t \sqsubseteq^E s.t{\restriction}A^+ = t{\restriction} A^-\}$. Thus we may form a category $\mathcal{G}$ in which objects are A-rooted tree arenas and morphisms from A to B are strategies on $A \to B$.

$\mathcal{G}$ has all small coproducts, given by the "disjoint union" of arenas, and an initial object, the empty arena $\mathbf{0}$, containing no moves. We define *premonoidal* structure on $\mathcal{G}$ based on that described in [5,13,11], in which initial moves of $A \odot B$ are pairs of initial moves from A and B, but non-initial moves are either from A or from B.

Definition 8. *From A-rooted tree arenas A_1, A_2, we form $A_1 \odot A_2$:*

- $M_{A_1\odot A_2} = \{(m, n) \in (M_{A_1} \times M^I_{A_2}) \cup (M^I_{A_1} \times M_{A_2}) \mid i \in \nu(m) \cap \nu(n) \Longrightarrow \alpha(m)_i = \alpha(n)_i)\}$,
- $\lambda_{A_1\odot A_2}(\langle m_1, m_2\rangle) = \lambda_{A_2}(m_2)$, *if* $m_1 \in M^I_{A_1}$,
  $\lambda_{A_1\odot A_2}(\langle m_1, m_2\rangle) = \lambda_{A_1}(m_1)$, *otherwise,*
- $(m_1, m_2) \vdash_{A_1\odot A_2} (n_1, n_2)$ *if* $m_1 = n_1 \in M^I_{A_1}$ *and* $m_2 \vdash_{A_2} n_2$ *or* $m_2 = n_2 \in M^I_{A_2}$ *and* $m_1 \vdash_{A_2} n_1$,
- $\pi \cdot \langle m, n\rangle = \langle \pi \cdot m, \pi \cdot n\rangle$,
- $\alpha(m, n)_i = \alpha^{A_1}(m)_i$, *if* $i \in \nu(m)$,
  $\alpha(m, n)_i = \alpha^{A_2}(n)_i$, *otherwise.*

Examples:

Equality Testing. For each object A, we have a strategy $\mathsf{eq} : ch(A) \odot ch(A) \to I + I$ which is supplied by Opponent with a pair of names and responds with $\mathrm{in_l}(*)$ if they are equal and $\mathrm{in_r}(*)$ otherwise: $\ker(\mathsf{eq}) = \{\langle i, i\rangle \mathrm{in_l}(*) \mid i \in \mathbb{N}\} \cup \{\langle i, j\rangle \mathrm{in_r}(*) \mid i, j \in \mathbb{N} \wedge i \neq j\}$.

Message Passing. For each object A, we have a strategy $\mathsf{send} : ch(A) \odot A \to \mathbf{0}$ which is supplied with a channel name i and an initial move m in A, plays it as an initial move $\langle i, m \rangle$ in the dependent arena for A, and then plays copycat between the explicit and dependent occurrences of A. (See Fig. 1.)

For each object A we define an endofunctor $\_ \odot A : \mathcal{G} \to \mathcal{G}$: given $\sigma : B \to C$, $\sigma \odot A : B \odot A \to C \odot A = \{s \in L_{B \odot A \to C \odot A} \mid s \backslash A, A \in \sigma \wedge s \restriction A, A \in \mathsf{id}_A \wedge P_\nu(s \backslash A, A) = P_\nu(s)\}$.

Proposition 1. *$(\mathcal{G}, I, \odot)$ is a symmetric premonoidal category.*

We now identify a category in which the premonoidal product is Cartesian.

Definition 9. *A sequence $qas \in L_{A \to B}$ is* single-threaded *if a answers q and:*

- *Player does not introduce any new names with the move a — i.e $P_\nu(qa) = \varnothing$.*
- *There is at most one move justified by a in s.*

A strategy σ is single-threaded if it is non-empty, every sequence of at least two moves in $\ker(\sigma)$ is single-threaded and $qa, qa' \in \ker(\sigma)$ implies $a = a'$.

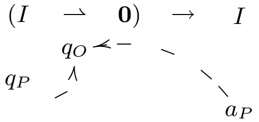
To define the composition of single-threaded strategies we apply a "promotion" operation $(\_)^\dagger$.

Definition 10. *Given a strategy $\sigma : A$, let $\sigma^\dagger$ be the least subset of L_A such that for any interaction sequence s, if $qa(s \restriction L) \in \sigma^\dagger$, $qa(s \restriction R) \in \sigma$ and $qa(s \restriction L \triangle R) \in L_A$ then $qa(s \restriction L \triangle R) \in \sigma^\dagger$.*

We define a category $\mathcal{G}_t$ with tree arenas as objects and single-threaded total strategies on $A \to B$ as morphisms from A to B. Composition of $\sigma : A \to B$ and $\tau : B \to C$ is defined $\sigma^\dagger; \tau$, and the identity on A is the restriction of $\mathsf{id}_A^{\mathcal{G}}$ to single-threaded sequences. We also note that $\_ \odot \_$ is a Cartesian product on $\mathcal{G}_t$. Thus $(\mathcal{G}, \mathcal{G}_t, (\_)^\dagger)$ is a Freyd category [16] (a symmetric premonoidal category $\mathcal{G}$, a Cartesian category $\mathcal{G}_t$, and an identity-on-objects strict symmetric premonoidal functor from $\mathcal{G}_t$ to $\mathcal{G}$). Moreover, it is a *closed* Freyd category: the functor $(\_)^\dagger \odot A : \mathcal{G}_t :\to \mathcal{G}$ has a right adjoint $A \rightharpoonup \_ : \mathcal{G} \to \mathcal{G}_t$.

Definition 11. *For any Q-rooted tree arena B, let $\uparrow B$ be the arena obtained by adding to B a single initial answer (invariant under G action) which enables all of the initial moves of B. We then define $A \rightharpoonup B = \uparrow (A \to B)$.*

Thus, for example, the arena $I \rightharpoonup \mathbf{0}$ consists of an initial answer which enables a single question. So we have a strategy $\mathsf{spawn} : (I \rightharpoonup \mathbf{0}) \to I$ which responds to the initial Opponent question by concurrently answering it and playing the (unique) initial question in $I \to \mathbf{0}$. i.e.:



There is an obvious bijection from (non-empty) legal sequences on $A \odot B \to C$ to single-threaded sequences on $A \to (B \rightharpoonup C)$, sending $\langle m, n\rangle s$ to $mans$ and yielding an adjunction between $A \rightharpoonup \_$ and $(\_)^\dagger \odot A : \mathcal{G}_t :\to \mathcal{G}$. Thus we have a model of the call-by-value λ-calculus, and so we may interpret terms $x_1 : S_1, \ldots, x_n : S_n \vdash M : T$ of $\mathcal{L}$ as morphisms from $\llbracket S_1 \rrbracket \odot \ldots \odot \llbracket S_n \rrbracket$ to $\llbracket T \rrbracket$ in $\mathcal{G}$ by setting $\llbracket \mathtt{0} \rrbracket = \mathbf{0}$, $\llbracket \mathtt{unit} \rrbracket = I$ and $\llbracket \mathtt{bool} \rrbracket = I + I$, $\llbracket S \Rightarrow T \rrbracket = \llbracket S \rrbracket \rightharpoonup \llbracket T \rrbracket$, and $\llbracket \mathtt{chan}[T] \rrbracket = ch(\llbracket T \rrbracket)$. The constants are interpreted as the key strategies already defined for channel-generation, equality testing, thread-spawning and sending and receiving values.

In order to prove soundness and adequacy with respect to may-testing (i.e. $M \Downarrow$ if and only if $M \neq \bot$) we define the interpretation of configurations $(\mathcal{T}, \mathcal{N})$. Given strategies $\sigma : A \to \mathtt{0}$ and $\tau : A \to B$, we define the asymmetric interleaving $\sigma \| \tau : A \to B$ to consist of sequences $qs \in L_{A \to B}$ such that there exists an interaction sequence t with $q(t \upharpoonright L) \in \sigma$, $q(t \upharpoonright R) \in \tau$ and $q(t \upharpoonright L \triangle R) = qs$. Then $f \| (g; h) = (f \| g); h$.

We interpret the configuration $M_1 : \mathtt{0}, \ldots, M_n : \mathtt{0}, N : S, \{a_1 : \mathtt{chan}[T_1], \ldots, a_m : \mathtt{chan}[T_n]\}$ as $\mathsf{new}_{\llbracket T_1 \rrbracket} \odot \ldots \mathsf{new}_{\llbracket T_m \rrbracket}; (\llbracket M_1 \rrbracket \| \ldots \| (\llbracket M_n \rrbracket \| \llbracket N \rrbracket))$.

Lemma 1. *If $C \longrightarrow C'$ then $\llbracket C \rrbracket \subseteq \llbracket C' \rrbracket$.*

Proof. We show that we may interpret evaluation contexts $a_1 : T_1, \ldots, a_n : T_n \vdash E[\cdot : S] : T$ as morphisms $\llbracket E[\_] \rrbracket : \llbracket S \rrbracket \odot \llbracket T_1 \rrbracket \odot \ldots \odot \llbracket T_n \rrbracket \to \llbracket T \rrbracket$ so that $\llbracket E[M] \rrbracket = \delta_{\llbracket T_1 \rrbracket \odot \ldots \odot \llbracket T_n \rrbracket}; (\llbracket M \rrbracket \odot (\llbracket T_1 \rrbracket \odot \ldots \odot \llbracket T_n \rrbracket)); \llbracket E[\_] \rrbracket$ and verify soundness for each reduction of the operational semantics using the categorical structure of $\mathcal{G}$ and the following (in)equations:

- For any $f : A \to \mathbf{0}$ and $g : I \to B$, $\Lambda(f); \mathsf{spawn}; g = f \| t_A; g$[1],
- $\pi_r \subseteq \mathsf{send} \| (\pi_l; \mathsf{recv})$,
- $\mathsf{new}_A; \langle \mathsf{id}_{ch(A)}, \mathsf{id}_{ch(A)}, \mathsf{id}_{ch(A)} \rangle^\dagger; \mathsf{eq} \odot ch(A) = \mathsf{new}_A; (\mathrm{in_l} \odot ch(A))$,
- $(\mathsf{new}_A \odot ch(A)); \langle \pi_l, \pi_r, \pi_l, \pi_r \rangle^\dagger; (\mathsf{eq} \odot (ch(A) \odot ch(A))) = (\mathsf{new}_A \odot ch(A)); \mathrm{in_r} \odot (ch(A) \odot ch(A))$.

Thus $M \Downarrow$ implies $M \neq \bot$.

Proposition 2. *If $\llbracket C \rrbracket \neq \bot$ then $M \Downarrow$.*

Proof. We define a translation which allows us to count internal reductions of M as $\mathtt{recv}$ actions: fixing a variable $c : \mathtt{chan}[\mathtt{unit}]$, for each term $\Gamma \vdash M : T$ with $c \notin \Gamma$, define $\Gamma, c \vdash M^c : T$:

- $M^c = M$ if M is a variable or constant.
- $(M\,N)^c = (\mathtt{recv}\,c); (M^c\,N^c)$
- $(\lambda x.M)^c = \lambda x.M^c$

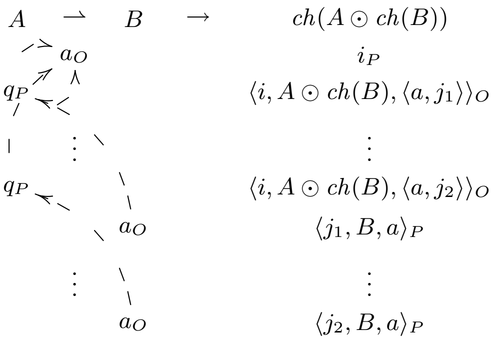
Then for every term $\Gamma \vdash M : T$, $\llbracket M \rrbracket = \llbracket \nu c.!\mathtt{send}\,(c, ()) | M^c \rrbracket$. Defining $\mathtt{send}^1\, M = \mathtt{send}\, M$ and $\mathtt{send}^{i+1}\, M = \mathtt{send}\, M | \mathtt{send}^i\, M$, we have $\llbracket !\mathtt{send}\, V \rrbracket = \bigcup_{i \in \mathbb{N}} \llbracket \mathtt{send}^i\, V \rrbracket$. So if $\llbracket M \rrbracket \neq \bot$ then by continuity, there exists n such that $\llbracket \mathtt{new} c.\mathtt{send}^n\, (c, ()) | M^c \rrbracket \neq \bot$. We may prove by induction on n that this entails that $M \Downarrow$, by showing that for any configuration $\llbracket C \rrbracket = \bigcup \{\llbracket C' \rrbracket \mid C \longrightarrow C'\}$.

[1] $t_A : A \to I$ is the terminal map in the category of single threaded strategies.

5 Definability and Full Abstraction

We prove full abstraction by establishing that for any legal sequence s over a type-object, the least strategy containing s (the closure of $\{s\}$ under the relations $\sqsubseteq$, $\sim$ and $\preceq$) is the denotation of a term. This is sufficient to define "tests" to distinguish any pair of distinct strategies.

Definition 12. *For any $s \in L_A^-$, let $\lceil s \rceil$ be the least set such that $s \in \lceil s \rceil$, and if $t \in \lceil s \rceil$ and $r \sqsubseteq t$ or $r \sim t$ or $r \preceq t$ then $r \in \lceil s \rceil$.*



$A \rightharpoonup B \rightarrow ch(A \odot ch(B))$
a_O
i_P
q_P
$\langle i, A \odot ch(B), \langle a, j_1 \rangle\rangle_O$
q_P
$\langle i, A \odot ch(B), \langle a, j_2 \rangle\rangle_O$
a_O
$\langle j_1, B, a \rangle_P$
a_O
$\langle j_2, B, a \rangle_P$

Fig. 2. A typical play of $[\![\mathtt{in}]\!](f)$

The key to our proof is the observation that we may encode justification pointers in terms of explicit name passing, and so reduce an arbitrary justfied sequence to one which is "pointer-free" (i.e. every move is an initial move). This reduction corresponds, in essence, to Sangiorgi's "triggering" translation of higher-order π-calculus into the π-calculus [18]. In the semantic setting, its essence may be captured as a *definable retraction* from the function type $S \Rightarrow T$ into the channel type $\mathtt{chan}[S * \mathtt{chan}[T]]$ (i.e. a pair of terms ($\mathtt{in} : (S \Rightarrow T) \Rightarrow \mathtt{chan}[S * \mathtt{chan}[T]], \mathtt{out} : \mathtt{chan}[S * \mathtt{chan}[T]] \Rightarrow S \Rightarrow T$) such that $[\![\lambda x.\mathtt{in}\,(\mathtt{out}\,x)]\!] = [\![\lambda x.x]\!]$.

Lemma 2. *For any types S, T there is a definable retraction from $S \Rightarrow T$ to $\mathtt{chan}[S * \mathtt{chan}[T]]$.*

Proof. We have $\mathtt{in} = \lambda f.\nu c.(\mathtt{let}\ z = \mathtt{recv}\ c\ \mathtt{in}\ \mathtt{send}\,(\mathtt{snd}(z), f\,\mathtt{fst}(z)))|c$, $\mathtt{out} = \lambda x.\lambda y.\nu d.\mathtt{send}\,(x, (y, d))|\mathtt{recv}\,d$.
There is a translation from legal plays of $[\![S \Rightarrow T]\!]$ to ($\sim$-equivalence classes of) legal plays of $[\![\mathtt{chan}[S * \mathtt{chan}[T]]\!]$, which adds arbitrary names i, j to the opening pair of moves, and replaces each move m with a justification pointer into one of these moves with a dependent move of the form $\langle i, m \rangle$ or $\langle j, m \rangle$. We prove that $\mathtt{in}$ and $\mathtt{out}$ act as copycats factoring through this translation (see Fig. 2).

We cannot use this retraction to reduce definability at all types to behaviour at the "π-types" constructed from B, $\_ * \_$ and $\mathtt{chan}[\_]$ (the problem is that $\mathtt{chan}[\_]$ is non-functorial and so "is a definable retract of" is not a precongruence). However, we may use it to map each strategy into one which has a "pointer-free fragment" from which the original strategy can be recovered. For each type T we define a type $\overline{T}$ as follows: $\overline{B} = B$; $\overline{S * T} = \overline{S} * \overline{T}$ and $\overline{S \Rightarrow T} = \mathtt{chan}[\overline{S} * \mathtt{chan}[\overline{T}]]$.

Proposition 3. *For each type T there is a definable retraction $(\mathtt{inj}_T, \mathtt{proj}_T) : T \trianglelefteq \overline{T}$ such that for all $s \in \llbracket T \rrbracket \to \mathbf{0}$ there exists a pointer-free $\overline{s} \in \llbracket x \vdash \mathtt{proj}_T\, x \rrbracket ; \lceil s \rceil$ such that $\llbracket x \vdash \mathtt{inj}_T\, x \rrbracket ; \lceil \overline{s} \rceil = \lceil s \rceil$.*

So given a sequence s in $\llbracket T \rrbracket \to \mathbf{0}$, if $\lceil \overline{s} \rceil$ is definable as a term $x : \overline{T} \vdash M : \mathtt{0}$ then $\lceil s \rceil$ is definable as $M\,(\mathtt{inj}_T\, x)$. We now show that each such pointer-free strategy is definable, via a decomposition which successively erases dependent moves.

Proposition 4. *For any pointer-free $s \in \llbracket T_1 \rrbracket \odot \ldots \odot \llbracket T_n \rrbracket \to \mathbf{0}$, $\lceil s \rceil$ is definable as a term $x_1 : T_1, \ldots, x_n : T_n \vdash M : \mathtt{0}$ such that $\llbracket M \rrbracket = \lceil s \rceil$.*

Proof. We assume $T_1, \ldots, T_n$ are *pointed* (i.e. base, function or channel types) and define M by induction on the length of s. If this is less than 2, then $\lceil s \rceil = \bot = \llbracket \mathtt{nil} \rrbracket$.

If s has length greater than 2 then $s = \langle a_1, \ldots, a_n \rangle \langle i, B, b \rangle s'$, where $i \in \nu(\langle a_1, \ldots, a_n \rangle)$ and thus $i = a_j$ for some $1 \leq j \leq n$, and so $B = \llbracket T_j \rrbracket$. So if $T_j = \mathtt{chan}[S_1 * \ldots * S_m]$ (where each S_k is pointed) then $b = \langle b_1, \ldots, b_m \rangle$ where $b_k \in M^I_{\llbracket S_k \rrbracket}$ for each k. So we may form a legal sequence $s'' = \langle a_1, \ldots, a_n, b_1, \ldots, b_m \rangle s'$ on $\llbracket T_n \rrbracket \odot \ldots \odot \llbracket T_n \rrbracket \odot \llbracket S_1 \rrbracket \odot \ldots \odot \llbracket S_m \rrbracket$. By induction hypothesis, $\lceil s'' \rceil$ is definable as a term $x_1 : T_1, \ldots, x_n : T_n, y_1 : S_1, \ldots, y_m : S_m \vdash M : \mathtt{0}$

If $\langle i, B, b \rangle$ is an Opponent move then we have $\lceil s \rceil = \llbracket \mathtt{let}\,(y_1, \ldots, y_m) = \mathtt{recv}\, x_i \;\mathtt{in}\; M \rrbracket$. If $\langle i, B, b \rangle$ is a Player move then for each $k \leq m$ we define a term $x_1 : T_1, \ldots, x_n : T_n \vdash N_k : S_k$:

- If $S_k = \mathtt{unit}$ then $N_k =_{df} ()$,
  If$S_k = \mathtt{bool}$ then $N_k =_{df} \mathtt{tt}$ if $b_k = \mathrm{in_l}(*)$ and $N_k =_{df} \mathtt{ff}$, otherwise.
- If $S_k = U \Rightarrow V$ then let $N_k =_{df} \lambda x.\mathtt{nil}$.
- If $S_k = \mathtt{chan}[U]$ then $N_k =_{df} x_i$, if $b_k = a_i$ and $b_k \neq x_j$ for $j < i$,
  $N_k =_{df} \mathtt{new}_u\,()$, if $b_k \neq x_j$ for all $j \leq n$.

For each $j \leq n$ we define a test term $B_j : \mathtt{bool}$:

- If $T_j = \mathtt{bool}$, we define $B_j =_{df} x_i$ if $a_j = \mathrm{in_l}(*)$ and $B_j =_{df} \neg x_i$, otherwise.
- If $T_j = U \Rightarrow V$ or $T_j = \mathtt{unit}$, then $B_j =_{df} \mathtt{tt}$,
- If $T_l = \mathtt{chan}[U]$ then $B_l = \bigwedge_{k \leq n} E_k$, where:
  $E_k =_{df} x_i = x_j$ if $T_j = T_k$ and $a_j = a_k$,
  $E_k =_{df} \neg x_j = x_k$ if $T_j = T_k$ and $a_j \neq a_k$,
  $E_k =_{df} \mathtt{tt}$, otherwise.

Then $\lceil s \rceil$ is definable as:
$\mathtt{let}\,(y_1, \ldots, y_m) = (N_1, \ldots, N_m)\;\; \mathtt{in\ If}\; \bigwedge_{j \leq n} B_j\; \mathtt{then}\,(\mathtt{send}\,(x_i, y_1, \ldots, y_m) | M)$
$\mathtt{else\ nil}$.

Theorem 1. *$\llbracket M \rrbracket \subseteq \llbracket N \rrbracket$ if and only if $M \lesssim N$.*

Proof. From right-to left (inequational soundness) this follows from soundness and adequacy. We prove the converse for closed values U, V, which implies the general case. So suppose $[\![U]\!] \not\subseteq [\![V]\!]$. Then there exists a sequence $qs \in I \rightarrow [\![T]\!]$ such that $qs \in [\![U]\!]$ and $qs \notin [\![V]\!]$. By Propositions 3 and 4, the strategy $\lceil s^{\perp} * \rceil$ on $[\![T]\!] \rightarrow \mathtt{unit}$ (where $*$ is the unique move in I) is definable as a term $x : T \vdash P : \mathtt{unit}$. Then $[\![(\lambda x.P)\, U]\!] = \{*\}$ and hence by adequacy, $(\lambda x.P)\, U \Downarrow$. But $[\![\lambda x.M\, V]\!] = \perp$, since for all $t* \in \lceil s^{\perp} * \rceil$ there exists $r \sim s^{\perp}$ such that $t \preceq r$ and hence $s \sim r^{\perp} \preceq t^{\perp}$ and so by assumption $qt^{\perp} \notin [\![V]\!]$. Hence $(\lambda x.M)\, V \not\Downarrow$, and $U \not\lesssim V$ as required.

References

1. W. Ferreira, M. Hennessy, and A. S. A. Jeffrey. A theory of weak bisimulation for core CML. *J. Functional Programming*, 8(5):447–491, 1998.
2. D. Ghica and G. McCusker. The regular language semantics of second-order Idealised Algol. *Theoretical Computer Science*, 309:469 – 502, 2003.
3. D. Ghica and A. Murawski. Angelic semantics of fine-grained concurrency. In *Proceedings of FOSSACS '04*, number 2987 in LNCS, pages 211–225. Springer, 2004.
4. R. Harmer and G. McCusker. A fully abstract games semantics for finite nondeterminism. In *Proceedings of LICS '99*. IEEE Computer Society Press, 1998.
5. K. Honda and N. Yoshida. Game theoretic analysis of call-by-value computation. In *Proceedings of ICALP '97*, volume 1256 of *LNCS*. Springer-Verlag, 1997.
6. D. Hughes. Games and definability for System F. In *Proceedings of LICS '97*. IEEE Computer Society Press, 1997.
7. J. M. E. Hyland and C.-H. L. Ong. On full abstraction for PCF: I, II and III. *Information and Computation*, 163:285–408, 2000.
8. A. S. A. Jeffrey and J. Rathke. Contextual equivalence for higher-order pi-calculus revisited. In *Logical Methods in Computer Science* 1(1:4), 2002.
9. A. S. A. Jeffrey and J. Rathke. A fully abstract may-testing semantics for concurrent objects. In *Proceedings of LICS '02*, pages 101–112, 2002.
10. J. Laird. A game semantics of ICSP. In *Proceedings of MFPS XVII*, number 45 in Electronic notes in Theoretical Computer Science. Elsevier, 2001.
11. J. Laird. A game semantics of names and pointers. To appear in Annals of Pure and Applied Logic, available from http://www.cogs.susx.ac.uk/users/jiml, 2005.
12. J. Laird. A game semantics of the asynchronous pi-calculus. In *Proceedings of CONCUR '05*, number 3653 in LNCS, pages 51–65. Springer, 2005.
13. O. Laurent. Polarized games. In *Proceedings of LICS '02*, 2002.
14. P. Malacaria and C. Hankin. Generalised flowcharts and games. In *Proceedings of ICALP '98*, 1998.
15. G. McCusker. *Games and full abstraction for a functional metalanguage with recursive types*. PhD thesis, Imperial College London, 1996.
16. J. Power and E. Robinson. Premonoidal categories and notions of computation. *Mathematical Structures in Computer Science*, 1997.
17. J. Reppy. CML: A higher-order concurrent language. In *Proceedings of PLDI '91*, pages 293–305. ACM Press, 1991.
18. D. Sangiorgi. *Expressing Mobility in Process Algebras: First-Order and Higher-Order Paradigms*. PhD thesis, University of Edinburgh, 1993.

Author Index