30th International Conference on Foundations of Software Technology and Theoretical Computer Science

FSTTCS 2010, December 15–18, 2010, Chennai, India

Edited by
Kamal Lodaya
Meena Mahajan

LIPIcs – Vol. 8 – FSTTCS 2010 www.dagstuhl.de/lipics

Editors

Kamal Lodaya and Meena Mahajan
The Institute of Mathematical Sciences
CIT Campus, Taramani
Chennai 600113 India
{kamal,meena}@imsc.res.in

ACM Classification 1998
D.2.4 Software/Program Verification, F.1.1 Models of Computation, F.1.2 Modes of Computation, F.1.3 Complexity Measures and Classes, F.2.2 Nonnumerical Algorithms and Problems, F.3.1 Specifying and Verifying and Reasoning about Program

ISBN 978-3-939897-23-1

Published online and open access by
Schloss Dagstuhl – Leibniz-Center for Informatics GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany.

Publication date
December, 2010.

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/LIPIcs.FSTTCS.2010.i

ISBN 978-3-939897-23-1 **ISSN 1868-8969** **http://www.dagstuhl.de/lipics**

LIPIcs – Leibniz International Proceedings in Informatics

LIPIcs is a series of high-quality conference proceedings across all fields in informatics. LIPIcs volumes are published according to the principle of Open Access, i.e., they are available online and free of charge.

ISSN 1868-8969

www.dagstuhl.de/lipics

Preface

This proceedings volume has the papers presented at the 30th annual conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010), held at the Institute of Mathematical Sciences (IMSc), Chennai, during 15–18 December 2010. We thank IMSc, in particular the administration, for their unstinted support in hosting this conference. Our colleagues and students pitched in to make the organization really smooth.

The Indian Association for Research in Computing Science (IARCS) grew out of the community built by this conference since its first edition in 1981 (the webpages `www.iarcs.org.in` and `www.fsttcs.org` have more details). FSTTCS is now the annual flagship IARCS event.

Over the years, the conference has been able to attract top-quality invited talks from renowned speakers all over the world, and this year is no exception. We thank Rajeev Alur, Bruno Courcelle, Pavel Pudlák, Santosh Vempala and Wiesław Zielonka for readily agreeing to come to FSTTCS, and for their contributions to this proceedings.

The conference attracted 128 submissions from 35 countries in 6 continents, most of them of very high quality. We thank the authors who submitted for making this such a competitive conference.

The difficult job of selecting from these submissions fell upon the Programme Committee (PC), where we had the pleasure of working with 22 eminent colleagues from 9 countries. We thank them all for increasing the stature of the conference with their presence, for the work they put in, and in some cases, for assisting us in resolution of tricky issues apart from determining just the technical merits of the paper. The PC succeeded in obtaining the help of 216 external reviewers, in all producing 400 referee reports which were of immeasurable help in deciding the 38 contributed papers which have made it to this publication. We thank the referees for their time, effort and detail.

The conference submission and proceedings preparation were handled on the popular Easychair software. We thank Andrei Voronkov of Easychair and our system administrators (especially Raveendra Reddy and Mangala Pandi) at IMSc for dealing with mysterious incompatibilities which surfaced occasionally.

The programme this year also features a special session on undergraduate and graduate curricula. We thank our colleague R. Ramanujam for stepping in with this idea to explore a new direction for the 30th conference, and following it through. The 5th international symposium on Parameterized and Exact Computation (IPEC) is co-located with our conference at Chennai, and is preceded by the IMPECS school on Parameterized and Exact Computation, and we are happy for the plurality this has added to the FSTTCS experience.

Since 2008, FSTTCS has been published in the LIPIcs series, which makes the contributions available free of cost on the web, and allows authors to retain their rights. We thank the LIPIcs board of editors and share the pride of being part of a high-quality series. Thanks also to Marc Herbstritt and his team who worked hard on the production side, coming up with the style file and patiently answering our never-ending questions!

Kamal Lodaya and Meena Mahajan
Chennai, December 2010

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: K. Lodaya, M. Mahajan

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Contents

Invited Talks

Contributed Papers

Session 1A

Session 1B

Session 2A

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: K. Lodaya, M. Mahajan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Session 2B

Session 3A

Session 3B

Session 4A

Session 4B

Session 5

Session 6A

Session 6B

Conference Organization

Programme Chairs

Kamal Lodaya
Meena Mahajan

Programme Committee

Sanjeev Arora *(Princeton U)*
Eike Best *(U Oldenburg)*
Ahmed Bouajjani *(LIAFA Paris)*
Amit Chakrabarti *(Dartmouth)*
Véronique Cortier *(LORIA-CNRS)*
Luca de Alfaro *(Google/UCSC)*
Xiaotie Deng *(CU Hong Kong)*
Khaled Elbassioni *(MPII Saarbrücken)*
Zoltán Ésik *(U Szeged)*
Fedor Fomin *(U Bergen)*
Martin Fürer *(Penn State U)*
Naveen Garg *(IIT Delhi)*
Joachim von zur Gathen *(U Bonn)*
Valentine Kabanets *(SFU Vancouver)*
T Kavitha *(TIFR Mumbai)*
P Madhusudan *(UIUC)*
Damian Niwiński *(U Warsaw)*
Prakash Panangaden *(McGill Montréal)*
Paritosh Pandya *(TIFR Mumbai)*
Günter Rote *(FU Berlin)*
Anil Seth *(IIT Kanpur)*
Wolfgang Thomas *(RWTH Aachen)*

Local Organization

K. Narayan Kumar *(CMI)*
Meena Mahajan *(IMSc) chair*
R. Ramanujam *(IMSc)*
Saket Saurabh *(IMSc)*
Vikram Sharma *(IMSc)*

External Reviewers

Parosh Abdulla
Klaus Aehlig
S. Arun-Kumar
Michael Backes
Andrew Badr
Christel Baier
Nikhil Bansal
Vince Barany
Pablo Barceló
Frederique Bassino
Shoham Ben-David
Nadja Betzler
Johannes Blömer
Vincenzo Bonifaci
Laurent Braud
Mark Braverman
Franck van Breugel
Tian-Ming Bu
Arnaud Carayol
Sourav Chakraborty
Krishnendu Chatterjee
Konstantinos Chatzikokolakis
Swarat Chaudhuri
Prasad Chebolu
Marsha Chechik
Jing Chen
Janka Chlebikova
Horatiu Cirstea
Thomas Colcombet
Graham Cormode
Bruno Courcelle
Ugo Dal Lago

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: K. Lodaya, M. Mahajan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Zhe Dang
Samir Datta
Aldric Degorre
Stephanie Delaune
Holger Dell
Stephane Demri
Amit Deshpande
Alin Deutsch
Laurent Doyen
Deepak D'Souza
Kunal Dutta
Laila El Aimani
Amr Elmasry
Leah Epstein
Uli Fahrenberg
Qizhi Fang
Mohamed Faouzi Atig
Arash Farzan
Henning Fernau
Carla Ferreira
Eldar Fischer
Hans Fleischhack
Vojtech Forejt
Martin Fränzle
Sibylle Froeschle
Ricard Gavalda
Cyril Gavoille
Blaise Genest
Sonja Georgievska
Panos Giannopoulos
Hugo Gimbert
Xavier Goaoc
Guillem Godoy
Petr Golovach
Valentin Goranko
Georg Gottlob
Natalya Gribovskaya
Colas Le Guernic
H. Peter Gumm
Peter Habermehl
Magnus M. Halldorsson
Avinatan Hassidim
Elad Hazan
Danny Hermelin
Yoram Hirshfeld
Jochen Hoenicke
Peter Höfner
Markus Holzer
Florian Horn
Pavel Hrubes
Chien-Chung Huang
Lucian Ilie
Szabolcs Ivan
Joanna Jedrzejowicz
Lukasz Kaiser
Bruce Kapron
Hrishikesh Karmarkar
Andreas Karrenbauer
Anna Kasprzik
Subhash Khot
Astrid Kiehn
Pekka Kilpeläinen
Hartmut Klauck
Johannes Koebler
Robert Koenig
Eryk Kopczynski
Martin Kot
Stefan Kratsch
Raghav Kulkarni
Michal Kunc
Orna Kupferman
Martin Kutrib
Marcel Kyas
Ranko Lazic
Sergueï Lenglet
Jerome Leroux
Xueliang Li
Nutan Limaye
Zhiming Liu
Christof Löding
Daniel Loebenberger
Pinyan Lu
Andreas Maletti
Dániel Marx
David Matei
Richard Mayr
Andrew McGregor
Pierre McKenzie
Dieter van Melkebeek
Jakub Michaliszyn
Zoltan Miklos
Vahab Mirrokni
Neeldhara Misra
Joseph Mitchell
Ankur Moitra
Madhavan Mukund

Wolfgang Mulzer
Filip Murlak
Simin Nadjm-Tehrani
Satyadev Nandkumar
K Narayan Kumar
Meghana Nasre
Michael Nüsken
Jan Obdrzalek
Alexander Okhotin
Lorenzo Orecchia
Elena Oshevskaya
Sang-il Oum
Gennaro Parlato
Pawel Parys
Soumya Paul
Seth Pettie
Ramchandra Phawade
Cédric Piette
Sophie Pinchinat
CK Poon
Sanjiva Prasad
M. Praveen
S P Suresh
Venkatesh R
Yona Raekow
Rajiv Raman
R. Ramanujam
Saurabh Ray
Benoit Razet
Igor Razgon
Uday Reddy
Eike Ritter
Adam Rogalewicz
Sasanka Roy
Wojciech Rytter
Prakash Saivasan
Neyire Deniz Sarier
Saket Saurabh
Christian Schaffner
Sven Schewe
Pranab Sen
Rocco Servedio
C. Seshadhri
Krishna Shankara Narayanan
JianFeng Si
Somnath Sikdar
David Steurer
Colin Stirling
Mani Swaminathan
Jean-Marc Talbot
Tony Tan
Igor Tarasyuk
Tino Teige
Mark Timmer
Ashish Tiwari
Szymon Torunczyk
Michael Ummels
Tarmo Uustalu
Sándor Vágvölgyi
Gregory Valiant
György Vaszil
Enrico Vicario
Björn Victor
Yngve Villanger
V Vinay
Irina Virbitskaite
Mahesh Viswanathan
Magnus Wahlström
Yajun Wang
Daniel Werner
Uwe Wolter
James Worrell
David Xiao
Shaofa Yang
Huiwen Yu
Sheng Yu
Gianluigi Zavattaro
Huaming Zhang
Jinshan Zhang
Shengyu Zhang
Konstantin Ziegler
Martin Zimmermann

and six anonymous reviewers

Author Index

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: K. Lodaya, M. Mahajan
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Expressiveness of streaming string transducers

Rajeev Alur[1] and Pavol Černý[2]

1 University of Pennsylvania
2 IST Austria

Abstract

Streaming string transducers [1] define (partial) functions from input strings to output strings. A streaming string transducer makes a single pass through the input string and uses a finite set of variables that range over strings from the output alphabet. At every step, the transducer processes an input symbol, and updates all the variables in parallel using assignments whose right-hand-sides are concatenations of output symbols and variables with the restriction that a variable can be used at most once in a right-hand-side expression. It has been shown that streaming string transducers operating on strings over infinite data domains are of interest in algorithmic verification of list-processing programs, as they lead to PSPACE decision procedures for checking pre/post conditions and for checking semantic equivalence, for a well-defined class of heap-manipulating programs. In order to understand the theoretical expressiveness of streaming transducers, we focus on streaming transducers processing strings over finite alphabets, given the existence of a robust and well-studied class of "regular" transductions for this case. Such regular transductions can be defined either by two-way deterministic finite-state transducers, or using a logical MSO-based characterization. Our main result is that the expressiveness of streaming string transducers coincides exactly with this class of regular transductions.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.1

1 Introduction

Deterministic finite-state automata are a canonical model for finite-state *acceptors* of strings, since many variations turn out to be equally expressive and the resulting class of regular languages enjoys a number of desirable theoretical properties. In this paper, we focus on *transducer* models that define (partial) functions from input strings to output strings. The most natural model for a finite-state transducer is a finite-state machine that, at each step, reads an input symbol and produces zero or more output symbols. If we restrict such a machine to read the input string only once from left to right, then the model is too restrictive: while "delete all a symbols" can be implemented, "delete all a symbols, if the input string contains a b symbol" cannot be implemented. However, the *two-way deterministic finite-state transducers* have appealing theoretical properties: the equivalence problem is decidable, they are expressively equivalent to MSO (monadic second-order logic) definable transductions, and this class of "regular" transductions is closed under operations such as sequential composition [2, 4, 3].

Recently, we proposed the model of *streaming string transducers* for algorithmic verification of single-pass list processing programs [1]. A streaming string transducer makes a single pass through the input string to produce an output string. It uses a finite set of variables that range over strings from the output alphabet. At every step, the transducer processes an input symbol, and updates all the variables in parallel using assignments whose right-hand-sides are concatenations of output symbols and variables with the restriction that a variable can be used at most once in a right-hand-side expression. For example, with two variables x and y, the update $(x, y) = (x.y, a)$ sets x to the concatenation of x and y, and sets y to the constant a. While such an update is permitted, the update $(x, y) = (x.y, y)$ is not, since

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 1–12

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

y appears twice in the right-hand-sides, and would amount to "copying". The output in a given state is specified as a concatenation of output symbols and variables with a similar no-copy restriction. Unlike classical tape-based models, the streaming string transducer is not constrained to add output symbols only at the end, and can compute the output in multiple chunks that can be extended and concatenated as needed.

Streaming string transducers have been shown to be useful for algorithmic verification of list processing programs [1]. The problems of checking functional equivalence of two streaming transducers, and of checking whether a streaming transducer satisfies pre/post verification conditions specified by streaming acceptors over input/output strings, are decidable with PSPACE complexity. There is an expressively equivalent class of imperative programs that manipulate heap-allocated single-linked list data structure, as well as a corresponding class of list-processing functional programs with syntactic restrictions on recursive calls. These results lead to algorithms for checking functional equivalence of two programs, written possibly in different programming styles, for commonly used routines such as insert, delete, and reverse.

A goal of this paper is to study the expressiveness of streaming string transducers in order to gain better theoretical insights into their computing power. The transducers in [1] process strings over a potentially infinite data domain that supports the operations of equality and ordering. Given that the notion of regular transductions is well-understood in the context of strings over finite alphabets, we restrict our attention to streaming transducers that process strings over finite alphabets. The main result of the paper is that streaming string transducers exactly capture regular transductions, and thus, are equivalent to two-way transducers as well as MSO-definable string transductions.

In order to develop our results, we consider another single-pass transducer model called *heap-based string transducer* that reads the input string from left to right in a single pass, and computes the output using a heap of cells each of which can store an output symbol and has a next pointer. The next-pointers induce an (unranked) forest structure over cells. The transducer accesses the heap using a finite number of pointer variables, and can change next-pointers of cells referenced by these variables. It can also add new cells to the heap. The sequence of symbols labeling the cells accessible from a state-dependent output-pointer is the output of the transducer. For example, to output the *reverse* of the input string, the heap-based transducer, at each step, reads the next input symbol, and adds a cell containing this input symbol to the front of the output list being computed, exactly the same way as a C program would reverse a linked-list in a single pass. While proving assertions of programs manipulating heaps is typically undecidable [6], the key restriction for our model of a heap-based transducer is that it can update, but not traverse, the next-pointers of the cells referenced by its pointer variables (that is, an assignment of the form $next(x) = y$ is allowed, but $x = next(y)$ is not). Heap-based transducers can be viewed as syntactically restricted and abstract version of imperative single-pass programs studied in [1].

We first show that given a two-way transducer, one can construct an equivalent one-way heap-based transducer. The proof builds on the classical simulation of a two-way acceptor by a one-way acceptor [7], but needs new insights in order to maintain the potentially needed output segments in a shared heap that can be modified using a bounded number of updates at each step. The fact that a streaming string transducer can simulate a heap-based transducer is a corollary of a result of [1] that shows that single-pass list processing programs can be simulated by streaming transducers. Finally, we establish that a streaming string transducer can be captured by an MSO-definable string transformation, thereby establishing equivalence of all the transducer models.

We also show that streaming string transducers are closed under sequential composition.

This result follows from the MSO characterization, but we give a direct proof using summaries of a computation of a streaming string transducer. Finally, we show that extending heap-based transducers by allowing the traversal instruction that updates a pointer to the next-pointer of a referenced cell, leads to a strictly more expressive model than the class of regular transductions.

2 Regular String Transductions

A *deterministic transduction* from an input alphabet Σ to an output alphabet Γ is a partial function from Σ^* to Γ^*. We briefly review two equivalent ways of defining deterministic transductions using two-way finite-state transducers and using monadic second-order logic (MSO), and some known results (the details can be found in [3]). We will use the following three transductions throughout the paper. Let $\Sigma = \Gamma = \{a, b\}$. The transduction f_1 rewrites an input string to the string followed by its reverse: $f_1(w) = w.rev(w)$. For the transduction f_2, if the input string ends with the letter b, then f_2 deletes all occurrences of a, otherwise it leaves the input unchanged: if $w \in \Sigma^* b$ then $f_2(w) = b^k$ where k is the number of b's in w, else $f_2(w) = w$. The transduction f_3 replaces each symbol b by as many b's as there are a's between this occurrence of b and the previous occurrence of b: $f_3(a^{i_1}ba^{i_2}b\cdots a^{i_k}ba^{i_{k+1}}) = a^{i_1}b^{i_1}a^{i_2}b^{i_2}\cdots a^{i_k}b^{i_k}a^{i_{k+1}}$.

A *two-way deterministic (finite-state) transducer* M from input alphabet Σ to output alphabet Γ consists of a finite set of states Q, an initial state $q_0 \in Q$, a final state $q_f \in Q$, and a transition function δ from $Q \times (\Sigma \cup \{\vdash, \dashv\})$ to $Q \times \{-1, 0, +1\} \times \Gamma^*$. The symbols $\vdash$ and $\dashv$ are used to mark the two ends of the input. Given an input string $w \in \Sigma^*$, the transducer M starts in state q_0 with the input tape containing the string $\vdash w \dashv$, scanning the left-most symbol. At every step, based on the current state q and the current input symbol a, the machine updates the state (as specified by the first component of $\delta(q, a)$), moves the read-head (as specified by the second component of $\delta(q, a)$, where -1 means move left, 0 means stay put, and $+1$ means move right), and outputs a sequence of symbols in Γ (as specified by the third component of $\delta(q, a)$). If the current state is the final state q_f, the machine stops, and in this case, the output $[\![M]\!](w)$ corresponding to the input string w is the concatenation of outputs emitted along the run. If the machine never enters the final state, or tries to move left while reading $\vdash$, or tries to move right while reading $\dashv$, then this is an error, and $[\![M]\!](w)$ is undefined. This partial function $[\![M]\!]$ defines the semantics of the machine M, and is a deterministic transduction from Σ to Γ.

To illustrate the definition of a two-way transducer, let us consider how to implement the example transductions by two-way transducers. To implement the transduction f_1, the two-way transducer reads the string in one left-to-right pass followed by one right-to-left pass, emitting the symbol read from the input at every step. The two-way transducer for f_2, first moves all the way to the right. If the last symbol is b, then while moving right to left it outputs b for every b symbol it reads, while ignoring a symbols. If the last symbol is not b, it moves all the way to the left, and using a final left-to-right pass, it outputs every symbol it reads. The transducer for f_3 starts moving to the right emitting an a symbol for each a symbol read. If it encounters the right end-marker, it halts, If it encounters a b symbol, it emits the empty string and starts moving left. For every a symbol it reads, it emits b, and keeps moving left. When it encounters a b symbol or the left end-marker, it starts moving right again skipping over the a's until it encounters a b, emitting the empty string at each step. Then the whole cycle repeats.

For defining transductions using monadic second-order logic, a string $w = w_1 w_2 \dots w_k$

is viewed as a (string) graph G_w with $k+1$ vertices $v_0 v_1 \dots v_k$, with an edge from each v_i to v_{i+1} labeled with the symbol w_i. Then, an MSO formula over an alphabet Σ, to be interpreted over such a graph G_w, consists of Boolean connectives, quantifiers, first-order variables that range over vertices of G_w, monadic second-order variables that range over sets of vertices of G_w, and atomic formulas of the form $a(x, y)$, for $a \in \Sigma$, meaning that the vertex x has an a-labeled edge to the vertex y. A *deterministic MSO transducer* T from input alphabet Σ to output alphabet Γ consists of a finite copy set C, vertex formulas $\varphi^c(x)$, for each $c \in C$, each of which is an MSO formula over Σ with one free first-order variable x, and edge formulas $\varphi_a^{c,d}(x, y)$, for each $a \in \Gamma$ and $c, d \in C$, each of which is an MSO formula over Σ with two free first-order variables x and y. Given an input string w, consider the following output graph: for each vertex x in G_w and $c \in C$, there is a vertex x^c in the output if the formula $\varphi^c(x)$ holds over G_w, and for all such vertices x^c and y^d, there is an a-labeled edge from x^c to y^d if the formula $\varphi_a^{c,d}(x, y)$ holds over G_w. If this graph is the string graph corresponding to the string u over Γ then $[\![T]\!](w) = u$, and if this graph is not a string graph, then $[\![T]\!](w)$ is undefined.

Let us revisit our example transductions. To define the transduction f_1, we choose the copy set $C = \{1, 2\}$. The output graph retains both copies of each vertex in input graph, except the last one. When $a(x, y)$ holds in the input graph $a(x^1, y^1)$ and $a(y^1, x^1)$ holds in the output graph (the last vertex needs to be handled specially to connect the two copies). For the transduction f_2, we need only one copy of each vertex. The formula φ^1 is defined so that $\varphi^1(x)$ holds precisely when either the position x corresponds to a b-symbol in the input graph, or if the last symbol in the input is b (such "regular" look-ahead is easily definable using MSO formulas). The output graph has an edge from x to y if $\varphi^1(x)$ holds, y is the least position following x for which $\varphi^1(y)$ holds, and the label is the same as the input symbol corresponding to position x. The transduction f_3 can be defined by a deterministic MSO transducer with the copy set $C = \{1, 2\}$, where both output copies of a vertex are retained, or omitted, depending on whether the corresponding symbol in the input string is a, or b, respectively. The edge formulas are slightly complicated, but can be defined in MSO [3].

The two frameworks are equally expressive [3], and this class of transductions is called *regular string transductions*. The class of regular transductions is closed under sequential composition [2], and it is decidable to check whether two such transductions, presented by, say, two two-way deterministic transducers, are equivalent [4]. Observe that for any regular transduction, the ratio of the length of the output string to the length of the input string is bounded by a constant, namely, the size of the copy set C of the corresponding deterministic MSO transducer.

3 Streaming Transducer Model

A *(deterministic) streaming string transducer* W from input alphabet Σ to output alphabet Γ consists of a finite set of states Q, an initial state $q_0 \in Q$, a finite set of variables X, a partial output function F from Q to $(\Gamma \cup X)^*$ such that for each $q \in Q$ and $x \in X$, there is at most one occurrence of x in $F(q)$, a state-transition function δ_1 from $Q \times \Sigma$ to Q, and a variable-update function δ_2 from $Q \times \Sigma \times X$ to $(\Gamma \cup X)^*$ such that for each $q \in Q$ and $a \in \Sigma$ and $x \in X$, there is at most one occurrence of x in the set of strings $\{\delta_2(q, a, y) \mid y \in X\}$.

To define the semantics of such a transducer, consider configurations of the form (q, s), where s is a valuation from X to Γ^*. A valuation from X to Γ^* is extended to a valuation from $(X \cup \Gamma)^*$ to Γ^* in the natural way. The initial configuration is (q_0, s_0) where s_0 maps each variable to the empty string. The transition function is defined by $\delta((q, s), a) = (\delta_1(q, a), s')$

where for each variable x, $s'(x) = s(\delta_2(q, a, x))$. For an input string $w \in \Sigma^*$, if $\delta^*((q_0, s_0), w) = (q, s)$, then if $F(q)$ is undefined then so is $[\![W]\!](w)$, otherwise $[\![W]\!](w) = s(F(q))$.

The transduction f_1 can be implemented by a streaming string transducer W_{rev} with a single state and two variables x and y. Each symbol a is processed by the update $(x, y) = (xa, ay)$, and the output function is xy.

The transduction f_2 can be implemented by a streaming string transducer with two states q_0 and q_1, and two variables x and y. Initially the state is q_0, and the transducer is in state q_1 precisely when the most recent input symbol is b (it goes to state q_1 on reading b and to state q_0 on reading a). At every step, x contains the input string read so far, and y contains only b's encountered so far (the variable-update function on reading a is $(x, y) = (xa, y)$, and on reading b is $(x, y) = (xb, yb)$). In state q_1 the output function returns y, in state q_0 the output function returns x.

The transduction f_3 can be implemented by a streaming string transducer W_{cp} with a single state and two variables x and y. The symbol a is processed by the update $(x, y) = (xa, yb)$, the symbol b is processed by $(x, y) = (xy, \varepsilon)$, and the output function is x.

The following proposition states that streaming string transducers are closed under sequential composition. Note that streaming data string transducers [1] are not closed under sequential composition.

▶ Proposition 1. Given a streaming string transducer W_1 from input alphabet Σ_1 to output alphabet Σ_2 and a streaming string transducer W_2 from input alphabet Σ_2 to output alphabet Σ_3, one can effectively construct a streaming string transducer W from input alphabet Σ_1 to output alphabet Σ_3, such that for all strings w in Σ_1^*, we have that $[\![W]\!](w) = [\![W_2]\!]([\![W_1]\!](w))$ if $[\![W_1]\!](w)$ and $[\![W_2]\!]([\![W_1]\!](w))$ are both defined, and $[\![W]\!](w)$ is undefined otherwise.

Proof. We first define a notion of a summary of a computation for a streaming string transducer on an input string w. We then show how the transducer W can compute the sequential composition by simulating W_1 and keeping track of summaries of W_2 corresponding to string variables of W_1.

Given a string w, a summary of a finite state automaton is just a pair of states (q, q'), indicating that if the automaton starts reading w in a state q, it finishes in state q'. For a deterministic streaming string transducer U from Σ to Γ, a summary for a given start state q and a string w includes not only an end state q', but also a *string variable summary*, that is, a description of how the contents of the string variables get updated while processing w. Let X be the set of string variables of U. A string variable summary is a function κ from X to strings in $(\Gamma \cup X)^*$, with the same restrictions of copyless assignments as in the definition of streaming string transducers. For every valuation s from X to Γ^*, if the configuration of U is (q, s), then after processing the input string w, the configuration of U is (q', s') where $s'(x) = s(\kappa(x))$. Note that the computation of U, and hence the summary, is not influenced by the valuation s of string variables at the start of the computation on w.

A key observation is that a string variable summary can be represented by a set X' of $2 \times k$ string variables, where k is the number of string variables in X, in addition to finite bookkeeping information which is also independent in size from w. For instance, if X consists of two variables, the function κ can be $\kappa(x) = \alpha x \beta y \gamma, \kappa(y) = \iota$, or $\kappa(x) = \alpha y \beta, \kappa(y) = \delta x \iota$, or a similar combination, where $\alpha, \beta, \gamma, \iota$ are strings over Γ. The summary can be represented by X' containing in this case four string variables (that will store $\alpha, \beta, \gamma, \iota$), and a bounded amount of bookkeeping information to store how κ is constructed from X and X'.

We now describe the construction of the streaming string transducer W that computes the sequential composition of W_1 and W_2. The transducer W simulates W_1 processing the input string, and for every string variable x of W_1, it maintains a summary of W_2. It is easy to see

how summaries for string variables of W_1 are maintained: for example, consider the case when W_1 executes an assignment $z = az_1z_2$. First, we construct a string variable summary for computation of W_2 on processing the letter a. Then we compose this summary with summaries for strings stored in string variables z_1 and z_2. We explain how string variable summaries are composed on the following example. Let κ_1 be a string variable summary of a computation of W_2 on z_1 defined by $\kappa_1(x) = x\alpha y, \kappa_1(y) = \beta$. Let κ_2 be a string variable summary of a computation of W_2 on z_2 defined by $\kappa_2(x) = \epsilon, \kappa_2(y) = x\gamma y\iota$. Note that each of the strings $\alpha, \beta, \gamma, \iota$ can be stored in a string variable. Then the string variable summary κ after processing z_1z_2 is defined by $\kappa_2(x) = \epsilon, \kappa_2(y) = x\alpha y\gamma\beta\iota$ (where the string $\gamma\beta\iota$ can be stored in one string variable). This finishes the construction. The number of string variables of W is $m_1 \times n_2 \times (2 \times m_2)$, where m_1 is the number of string variables of W_1, n_2 is the number of states of W_2 and m_2 is the number of string variables of n_2. ◀

As the following proposition shows, streaming string transducers are closed under conditional composition, where the condition is given as a regular language over the input alphabet. The proof is based on product construction and is similar to the proof of closure under conditional composition for streaming data string transducers [1].

▶ Proposition 2. Given two streaming string transducers W_1 and W_2 from input alphabet Σ to output alphabet Γ and a regular language L over Σ, there exists a streaming stream transducer W such that for all strings $w \in \Sigma^*$, (i) if $w \in L$, then $[\![W]\!](w) = [\![W_1]\!](w)$ if $[\![W_1]\!](w)$ is defined and is undefined otherwise, and (ii) if $w \notin L$, then $[\![W]\!](w) = [\![W_2]\!](w)$ if $[\![W_2]\!](w)$ is defined, and is undefined otherwise.

4 Heap-based Transducer Model

A *heap-based (finite-state) deterministic string transducer* H from input alphabet Σ to output alphabet Γ consists of a finite set of states Q, an initial state $q_0 \in Q$, a finite set of pointers X, an output function F from states Q to variables X, and a transition function δ from $Q \times \Sigma$ to $Q \times A_X^*$, where the set A_X of atomic actions consists of $x := \nu(a)$ and $\eta(x) := y$, for $x, y \in X$ and $a \in \Gamma$.

Given an input string over Σ, the transducer computes the output by maintaining a heap. A *heap* h consists of a finite set C of cells, a mapping ℓ from C to Γ ($\ell(c)$ denotes the output symbol stored in the cell c), a mapping η from C to $C_\perp$, where $C_\perp$ denotes the set $C \cup \{\perp\}$ ($\eta(c)$ denotes the cell that the next-pointer of the cell c points to, with $\perp$ denoting the null value), and a mapping μ from X to $C_\perp$ ($\mu(x)$ is the cell that the pointer x points to). A configuration of the transducer H consists of a state $q \in Q$ and a heap h. Initially, the state is q_0 and in the initial heap h_0, C is empty and $\mu(x) = \perp$ for each x. The action $x := \nu(a)$ creates a new cell labeled by a symbol $a \in \Gamma$ and makes x point to the new cell. The action $\eta(x) := y$ modifies the next pointer of the cell pointed to by x to make it point to the cell pointed to by y, if x is not nil; if x is nil, the action has no effect. More formally, each action in A_X updates the heap as follows: $(C, \ell, \eta, \mu) \xrightarrow{x:=\nu(a)} (C', \ell', \eta', \mu)$ holds where $C' = C \cup \{c\}$, with $c \notin C$ [1], $\ell'(c) = a$, $\eta'(c) = \perp$ and ℓ' and μ' agree with ℓ and μ, respectively, on cells in C; $(C, \ell, \eta, \mu) \xrightarrow{\eta(x):=y} (C, \ell, \eta', \mu)$ holds where if $\mu(x) = c$ then $\eta'(c) = \mu(y)$ and η' agrees with η for cells other than c (if $\mu(x) = \perp$, the action has no effect). This transition relation can be lifted to configurations: for an input symbol $a \in \Sigma$, $(q, h) \xrightarrow{a} (q', h')$ if $\delta(q, a) = (q', \alpha)$ and

[1] The behavior of the transducer is deterministic as long as the choice of this new cell c is according to some deterministic naming policy.

$h \xrightarrow{\alpha} h'$. For an input string w, if $(q_0, h_0) \xrightarrow{w} (q, h)$ then h is the output heap corresponding to w. In the output heap $h = (C, \ell, \eta, \mu)$, if $\mu(F(q)) = \perp$ then $[\![H]\!](w)$ is the empty string. If the sequence of next-pointers starting from the cell $\mu(F(q))$ contains a cycle, then $[\![H]\!](w)$ is undefined. Otherwise, let $c_0 c_1 \dots c_n$ be the unique sequence of cells such that $c_0 = \mu(F(q))$, $\eta(c_i) = c_{i+1}$ for $i < n$, and $\eta(c_n) = \perp$. Then $[\![H]\!](w)$ is defined to be $\ell(c_0)\ell(c_1)\cdots\ell(c_n)$.

Such a transducer can implement the transduction f_1 as follows. After processing an input string w, suppose the current heap is a linear sequence of cells storing $w.rev(w)$ such that the pointer x_0 points to the first cell, the pointer x_1 points to the last cell corresponding to w and x_2 points to the first cell holding $rev(w)$, with an additional auxiliary pointer x_3. When the transducer reads the next input symbol, say a, it adds two new a-labeled cells in the middle of the current output, by executing the sequence $x_3 := \nu(a); \eta(x_1) := x_3; x_1 := \nu(a); \eta(x_3) := x_1; \eta(x_1) := x_2$. In the updated heap, x_3 and x_1 point to the two middle cells, and x_2 can be used as an auxiliary pointer. The result is given by the output pointer x_0.

To implement f_2, the heap-based transducer maintains two lists with pointers to the two ends of both the lists. An input symbol a is added to the end of only the first list by creating one new a-labeled cell, while an input symbol b is added to the end of both the lists by creating two new b-labeled cells. The transducer needs two states, and the state remembers whether the last symbol is b or not, and the state is used to return the pointer that points to the head of the appropriately chosen list.

To implement f_3, the heap-based transducer again maintains two lists with pointers to the two ends of both the lists. The output pointer points to the head of the first list. To process the input symbol a, it adds a a-labeled cell at the end of the first list, and a b-labeled cell at the end of the second list. To process the input symbol b, the first list is updated to the concatenation of the two (which can be implemented by changing the next-pointer of the last cell of the first list point to the first cell of the second list) and the second list is set to empty.

The two types of pointer manipulating instructions (node creation and next-pointer modification) are expressively adequate for our purpose. The class of programs in [1] allows a more general set of instructions for manipulating pointer variables (such as testing whether two pointers point to the same cell, testing whether a pointer is null, checking and updating the symbol stored at a cell pointed to by a pointer, assigning one pointer to another). From the results of this paper and the compilation of such programs into streaming transducers described in [1], it follows that such extensions do not add to expressiveness. The key missing instruction is the *traversal assignment* $x := \nu(y)$. Consider the transduction *merge*: given an input $u_1 u_2 \dots u_m \# v_1 v_2 \dots v_m$, output $u_1 v_1 u_2 v_2 \dots u_m v_m$. Adding traversal assignments would allow the heap-based transducer to define the *merge* transduction by using two traversal pointers, one traversing the u part of the input and the other traversing the v part of the input. The next proposition establishes that this transduction cannot be captured by a streaming string transducer. As we show later in this paper that heap-based string transducers and streaming string transducers exactly define regular transductions, the proposition implies that adding the traversal assignment strictly increases the expressiveness of heap-based string transducers.

▶ Proposition 3. The transduction *merge* is not definable by a streaming stream transducers.

Proof. Let us consider a streaming string transducer W with n states and k string variables and let us assume that W defines *merge*. We derive a contradiction as follows. Consider the set of inputs I_m where m is the length of the sequence $u = u_1 u_2 \dots u_m$ (resp. $v = v_1 v_2 \dots v_m$), and where u_i is in $\{a, b\}$ for all i such that $1 \leq i \leq m$, and v_j is in $\{c, d\}$, for all j such that $1 \leq j \leq m$.

Intuitively, the proof is simple: if x_1,..., x_k are values of string variables after reading the first part of the input, $u\#$, then these will appear (each at most once) as substrings in the final output. Assuming that $x_1, \ldots, x_k$ contain only a's and b's after processing $u\#$, and that while processing the second part of the input, v, W adds only c's and d's to the string variables, it is clear that the final output can contain at most a bounded number of alternations of letters in $\{a, b\}$ with letters in $\{c, d\}$. The transducer W therefore does not implement *merge*. We now formalize this argument.

Let us consider the set of inputs I_m as above, with m such that $2^m > n * k * 2^r$, where r is such that $2^r > k$. A string w is called *short* if $|w| \leq r$. Let us call a *short-configuration* of W the pair (q, ρ) where q is a state of W and ρ is a valuation of string variables, where each variable is assigned either a short string or a $*$. A short-configuration is an abstraction of a configuration of W, where a variable x keeps its original value if its value is a short string, otherwise it is abstracted by $*$. If we prove the following claim, we have that W does not define *merge*, and we obtain the desired contradiction.

Claim: There exist two different strings w_1 or w_2 from I_m such that (i) either the output on both w_1 and w_2 is the same, (ii) or the output on one of the strings w_1 and w_2 is incorrect.

To prove the claim, consider two different strings u_1 and u_2 in $\{a, b\}^m$ such that that W is in the same short configuration after processing u_1 and u_2. Such strings exist, as 2^m is more than the number of short configurations ($n \times k \times 2^r$). We now construct a string v in $\{c, d\}^m$ such that the claim holds for $u_1\#v$ and $u_2\#v$. Let us look at non-short strings that W stores after processing u_1 or u_2. If these are to be used in the output, they have to have c or d on every other position. These strings thus contain guesses for the v part of the whole input string. As the length of these strings is greater than r, W cannot have stored all the possible guesses for v (as it has k string variables, and $2^r > k$). We consider v that does not contain any sequence contained in a non-short string stored by W. Thus for $u_1\#v$ and $u_2\#v$ we have that if W uses in the output a string variable with a non-short string stored after processing u_1 (or u_2), then the output is incorrect. If it does not use any such string variable, the output will be the same in both cases (as W is deterministic). ◀

5 Expressiveness

5.1 From Two-way Transducers to Heap-based Transducers

▶ **Theorem 1.** *For every two-way deterministic transducer M with n states, there exists a deterministic heap-based transducer H with $O(n^n)$ states and $O(n)$ pointer variables such that $[\![M]\!] = [\![H]\!]$.*

Proof: The proof is an extension of Shepherdson's proof [7] for standard two-way finite automata to the case of transducers. We start by describing the main ideas of the proof. The heap-based transducer has only a single (left-to-right) pass on the input string. It thus needs to precompute information on possible back-and-forth traversals. More precisely, at each position p of w and for each state u of M, the information needed by H is captured by a pair (q_u, w_u) such that

if M reads the symbol at position p in state u, the first time M reaches the position $p+1$, it will be at state q_u, having produced output w_u along this stretch. We show how such pair can be updated, even in the presence of left moves. Let us suppose that the symbol at position p is the symbol a and that in state u, M moves left to state v and outputs string α. Furthermore, let us assume that for state v, a pair (q_v, w_v) was stored in the previous step; and that in state q_v the machine M moves right on a to q_u, outputting β. Then for state u, we need to store the pair $(q_u, \alpha w_v \beta)$. The situation is depicted in Figure 1.

We need to add a few important details to this intuitive explanation. First, H can store strings w_u (possible partial outputs of M) only on the heap. In order to store a string w_u, it will store two pointers, x^b_u and x^e_u. The two pointers will point to the first and last position of the string. Second, H cannot copy arbitrarily long strings, but it can instead use the fact that suffixes of strings represented on the heap can be shared. Third, in order to model the first forward traversal of M, we add a state m to the set of states of M. For each position p the pair (q_m, w_m) represents the situation that the first time M reaches $p+1$, it will be in state q_m and the output will be w_m. Fourth, it is possible that M never reaches the position $p+1$, either because it reached a final state and stopped, or because an error occurred, for example if M tried to move left on $\vdash$. If, for example, after a left move from a state u at position p, M reaches the final state q_f (and outputs w) before arriving at $p+1$, H represents this by the pair (q_f, w). On the other hand, if an error occurred, then H represents this by a pair (q_{err}, ϵ), where q_{err} is a special state. Fifth, note that H (as opposed to M) does not see the symbols $\vdash, \dashv$. This can be remedied by H having two copies of possible outputs of M as described above, with the additional copy assuming that the next character is $\dashv$ and simulating M on this symbol. The final output is the string starting at x, with x being the pointer variable representing the first forward traversal of M in the second copy of the possible outputs of M.

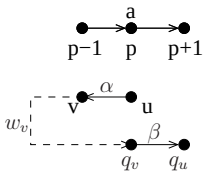


Figure 1 Representing the computation of M

We now present the construction in more details. The heap-based transducer H that we construct uses pointer-assignment instructions of the form $x = y$, where x and y are pointer variables. Such a heap-based transducer can be easily converted to a heap-based transducer H' which does not use such instructions. The state of H' records, in addition to the state of H, for every pair of pointers whether they are equal, and for every class such equal pointers, which pointer accesses the heap content. Then, to simulate the pointer assignment instruction $x = y$ in H, H' does not update x, but simply records that x and y are equal, and the pointer y points to the desired content. The construction does not change the number of pointers, but blows up the number of states by a factor of the number of possible partitions of the pointers. Since H has pointer assignment instructions, it suffices for H to have a single pointer variable x_o to be the output pointer in every state. The output function of H' maps a state to the pointer that is equal to x_o and accesses the actual content.

Let M be defined by the tuple $(\Sigma, \Gamma, Q_M, q^M_0, \delta_M)$. The heap-based transducer H is defined by the tuple $(\Sigma, \Gamma, Q_H, q^H_0, X, x^{b2}_m, \delta_H)$. Let Q_H be the set of functions from $Q_M \cup \{m\}$ to $Q_M \cup \{m, q_{err}\}$. Initial state is one which represents the identity function. The set of pointers contains four pointers x^b_u, x^e_u, x^{b2}_u, x^{e2}_u for each state u in $Q_M \cup \{m\}$. (The pointers x^{b2}_u, x^{e2}_u point to the beginning and end of the second copy of w_u.)

We explain the definition of the transition function δ_H using an example. Let us consider a state g of H such that $g(v) = q_v$ and a transition $\delta_M(u, a) = (v, -1, \alpha)$ in δ_M. Let us suppose that $\delta_M(q_v, a) = (q_u, +1, \beta)$. Then $\delta_H(g, a) = (f, A^*)$, for some state f such that $f(u) = q_u$. Recall that H stores the string w_v that M produced while moving from v to q_v on the heap, between pointers x^b_v and x^e_v. We have that while moving from u to q_u, M produces $\alpha w_v \beta$. The sequence of actions A^* puts α on the heap, make x^b_u point to the first node of α, and connects the last node of α to the node pointed to by x^b_v. Similarly, β is appended at x^e_v. So far, we have assumed that M moves right from q_v on the symbol a. If it moves left, we can again use the the information that H stores in the finite-state control and on the heap to find the state in which it will return to the current position and the string it

will output in the process. Note that the process may repeat several times, and potentially even cycle. However, H has all the necessary information stored locally. If it discovers a cycle (this happens for example if $q_v = u$), then the value of $f(u)$ will be q_{err}.

We describe how copying arbitrarily long strings on the heap is avoided by exploiting the fact that M is deterministic and the fact that suffixes of strings on the heap can be shared. Let us suppose that there are two states u and t of M such that M moves left on a symbol a to v from both u and t. Let us also suppose that at the previous step, H stored the information that from v, the first time H will move to the right it will be in state q_v and produce w_v in between. Using the idea described above, we obtain that we need to store the pair $(q_u, \alpha_u w_v \beta_u)$ for the state u, and the pair $(q_t, \alpha_t w_v \beta_t)$ for the state t. As the string w_v cannot be copied, it must be shared. As the heap is singly-linked, this would not be possible if β_u was different from β_t. However, since M is deterministic, we have that the execution from v will be identical in both cases — we have that $\beta_u = \beta_t$ (and $q_u = q_t$).

5.2 From Heap-based Transducers to Streaming Transducers

▶ **Theorem 2.** *For every deterministic heap-based string transducer H with n states and m pointer variables, there exists a deterministic streaming transducer W with $O(n2^m)$ states and $O(n)$ string variables such that $[\![H]\!] = [\![W]\!]$.*

Proof: The heap-based transducer and the streaming transducer both traverse the string only once. The difference between the two models is that one uses the heap, which allows sharing of suffixes of represented strings, whereas the other uses string variables, and thus sharing is not possible. The proof is the same as the compilation from imperative single-pass list-processing programs into streaming transducers described in [1], we describe it here for the sake of completeness.

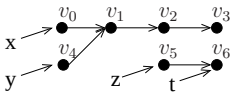


Figure 2 Singly-linked heap

The heart of the proof is therefore to show that the streaming transducer can represent the heap of the heap-based transducer using a bounded number of string variables. In order to achieve this, we adapt the approach (and terminology) of [5] for representing a singly-linked heap. A node v is called an *interruption* if it is either pointed to by a program variable or there are at least two distinct nodes with edges to v. An *uninterrupted list segment* is a finite sequence of nodes where: (i) the first node is an interruption, (ii) the next pointer of each node (except the last) points to the next node in the sequence, (iii) the last node is either an interruption or the value of its next pointer is nil, and (iv) no other node is an interruption. Consider the heap in Figure 2. The nodes v_0, v_1, v_4, v_5, v_6 are interruptions. The sequence of nodes $v_1 v_2 v_3$ is an uninterrupted list segment. It can be easily seen that for a heap-based transducer with k pointer variables, there can be at most $2k - 1$ interruptions and $2k - 1$ uninterrupted list segments. The heap can be compressed by replacing uninterrupted list segments by strings (to be stored in string variables of W).

The number of compressed heaps is exponential in the number of interruptions, and thus exponential in the number of pointer variables of H. The heap can therefore be stored by the streaming automaton W as follows: each uninterrupted list segment will be represented by a string variable. The shape of the heap (a forest) and the information on where the pointer variables of H point will be represented in the finite-state control of W. The number of states of W will be therefore linear in the number of states of H and exponential in the number of pointer variables of H.

It remains to show that the semantics of each instruction of the heap-based transducer can be modeled effectively on the representation described above. Let us consider the heap in Figure 2 and the action $\eta(y) := x$. The shape of the heap changes in two ways: first, the node pointed to by y points to the node pointed to by x (this information is kept in the finite-state control of W); second, there will be an uninterrupted list segment starting at the node pointed to by x containing the list segment $v_0v_1v_2v_3$. The (labels of nodes in) this list segment will be stored in a string variable. The string variable that in the previous step represented the list segment $v_1v_2v_3$ can be freed (and reused).

5.3 From Streaming Transducers to MSO

▶ **Theorem 3.** *For every deterministic streaming string transducer W there exists a deterministic MSO transducer T such that $[\![W]\!] = [\![T]\!]$.*

Proof. The proof is based on the idea that the computation of the streaming string transducer W can be represented in the copy set (from the definition of the MSO transducer). The unique sequence of states of W over a given input string w can be captured in MSO using second order existential quantification. Given a state of W and a letter in Σ, the function δ_2 is a function from the string variables to sequences in $(\Gamma \cup X)^*$. This function is what is represented using the copy set. The last step then consists of "reading out" the final output string from this representation.

We explain the encoding on an example. Let us consider the transduction f_1 and the streaming transducer W_{rev} defined in Section 3. The top of Figure 3 contains the input string *abb*. Recall that W_{rev} has only one state. The columns below each letter of the input string represent the assignments to the string variables: the top three nodes in a column represent the right-hand side (RHS) of the assignment $x := xa$ (resp. $x := xb$), the bottom three nodes represent the RHS of the assignment $y := ay$ (resp. $y := by$). The representation of an RHS with two symbols has three nodes. A symbol in Γ is represented by a marked edge, a variable z is represented by two nodes. The first of these nodes links to the previous column to where the representation of z begins. The second of these two nodes will be linked to from the last node representing z from the previous column. Such "linking" edges are represented by dashed (ε) lines in Figure 3.

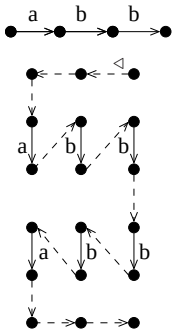


Figure 3 Representing the computation of W

In the first column, we connect the two nodes representing a variable in a right-hand side by a dashed (ε) line, as the string variables are initially empty. The last column is not used (and not pictured in Figure 3). In the column before last (the column that correspond to the last character of the input string), we also represent the effect of the output function $F(q_0) = xy$. We mark the first edge of x by a special symbol $\triangleleft$ to indicate where the output starts, and we connect the last symbol of x to the first symbol of y.

Note that the graph representing the computation of W_{rev} is not a string graph (there are disconnected nodes in the last column, not pictured), and it contains ε edges. We thus need to use an MSO transducer that deletes unused nodes (nodes unreachable from the marked node) and removes ε edges. Such a transducer is easily defined. We can then use the result that MSO transducers are closed under sequential composition [2] to conclude.

Details of the concrete MSO formulas are straightforward. We only note that the copy set of T needs to represent the right-hand side of any possible assignment to a string variable of

W, therefore (the upper bound on) the size of the copy set is $n * d$, where n is the number of pointer variables of W and d is the maximal length of the string $\delta(q, a, x)$, over all q (states of W), $a \in \Sigma$ and x (string variables of W).

6 Conclusions

The model of streaming string transducers is of potential interest for algorithmic verification of list-processing programs due to decidability of equivalence problem and correspondence to restricted classes of imperative heap-manipulating programs [1]. In this paper, we have established that its expressiveness coincides with the classical model of two-way transducers. This suggests robustness of their computing power. It also justifies the choice of primitive instructions in the corresponding class of heap-based transducers, which model single-pass programs that transform the input list using updates to a "traversal-free" heap consisting of singly-linked cells. A number of theoretical directions are worth pursuing. These include minimization of streaming string transducers, learning such transducers from input-output examples, synthesis of streaming transducers, extension to nondeterministic transducers, and extension to streaming transducers for tree-structured data.

References

1 R. Alur and P. Černý. Streaming transducers for algorithmic verification of single-pass list processing programs. In *Proceedings of the 38th Annual ACM Symposium on Principles of Programming Languages*, 2011.

2 M. Chytil and V. Jákl. Serial composition of 2-way finite-state transducers and simple programs on strings. In *Automata, Languages and Programming: Proceedings of Fourth International Colloquium, ICALP'77*, LNCS 52, pages 135–147. Springer, 1977.

3 J. Engelfriet and H. Hoogeboom. MSO definable string transductions and two-way finite-state transducers. *ACM Trans. Comput. Log.*, 2(2):216–254, 2001.

4 E. Gurari. The equivalence problem for deterministic two-way sequential transducers is decidable. *SIAM J. Comput.*, 11(3):448–452, 1982.

5 R. Manevich, E. Yahav, G. Ramalingam, and S. Sagiv. Predicate abstraction and canonical abstraction for singly-linked lists. In *Verification, Model Checking, and Abstract Interpretation, 6th International Conference*, LNCS 3385, pages 181–198. Springer, 2005.

6 G. Ramalingam. The undecidability of aliasing. *ACM Trans. Program. Lang. Syst.*, 16(5):1467–1471, 1994.

7 J. Shepherdson. The reduction of two-way automata to one-way automata. *IBM Journal of Research and Development*, 3:198–200, 1959.

Special tree-width and the verification of monadic second-order graph properties

Bruno Courcelle[1]

1 **Institut Universitaire de France**
Bordeaux University and LaBRI (CNRS)
F-33405, Talence, France
courcell@labri.fr

Abstract

The model-checking problem for *monadic second-order logic* on graphs is *fixed-parameter tractable* with respect to tree-width and clique-width. The proof constructs finite deterministic automata from monadic second-order sentences, but this produces automata of hyper-exponential sizes, and this computation is not avoidable. To overcome this difficulty, we propose to consider particular monadic second-order graph properties that are nevertheless interesting for Graph Theory and to *interpret automata* instead of trying to compile them (joint work with I. Durand).

For checking monadic second-order sentences written with *edge set quantifications*, the appropriate parameter is tree-width. We introduce *special tree-width*, a graph complexity measure between path-width and tree-width. The corresponding automata are easier to construct than those for tree-width.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.13

1 Introduction

It is well-known from [9,12,14] that the model-checking problem for monadic second-order (MS) logic on graphs is fixed-parameter tractable (FPT) with respect to tree-width and clique-width. The proof uses two main notions. First tree-decompositions (for tree-width as parameter) and k-expressions (for clique-width as parameter), and second, constructions of finite deterministic automata from the MS sentences that express the properties to check. These constructions use inductions on the structure of sentences. "Small" deterministic automata are built for atomic formulas. Conjunction and disjunction are reflected by products of automata. Existential quantification is easy but it introduces nondeterminism. Universal quantifications are replaced by negations and existential quantifications. Negation is reflected by complementation hence is easy on deterministic automata, but since existential quantifications produce nondeterminism, determinization must be performed before each application a complementation and this is the source of the hyper-exponential sizes of the constructed automata.

Two difficulties arise. Although the fact that a graph has tree-width at most k can be checked in linear time, the corresponding algorithm (by Bodlaender, see [12]) is not practically usable. The situation is even more difficult for clique-width (see [20] or Chapter 6 of [6]). One can argue that graphs are frequently given *with* decompositions witnessing that their tree-width or clique-width is at most some fixed k, but another difficulty arises : the automata to be constructed are extremely large and their computations run out of memory space. This is actually unavoidable if one wants algorithms taking as input any MS sentence (see, e.g., [15, 21, 22]). One possibility is to forget the idea of implementing the general theorem, and to work only on particular problems, as in [16-19] but we do not follow this direction: we present some techniques that can improve the situation in many significant cases.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 13–29

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

First, we limit our constructions to a fragment of MS logic whose sentences have limited alternations of quantifiers but that has nevertheless an interesting expressive power. Using Boolean set terms also helps to limit quantifier alternation and does not cost much in terms of sizes of automata. Second, we define deterministic automata for some basic graph properties and not only for the atomic formulas as is done usually in the proof of the general construction. Third, we *do not compile automata*, but instead we recompute their transitions whenever they are needed. Determinization is done "on the fly". These three ideas arise from joint work with I.Durand ([8]). Together with the necessary background notions, they will be presented in Sections 2 and 3 for graphs of bounded clique-width.

In Sections 4 to 6, we will explain how these constructions can be adapted, for graphs of bounded tree-width, to the verification of MS properties expressed with edge set quantifications. It appears that the corresponding automata, even for the basic property of adjacency, have exponential size in the bound on tree-width. This exponential blow-up does not occur if one uses path-decompositions instead of tree-decompositions. This observation motivates the introduction of *special tree-width*, a graph complexity measure intermediate between path-width and tree-width, and for which the basic automata are no more difficult to construct (or to specify) than for graphs of bounded clique-width. We will present some results that enlighten the differences between tree-width and special tree-width.

In this communication, we present the ideas and the main results. Technical details and proofs can be found in Chapter 6 of [6] and in [7].

2 Clique-width

Graphs are finite and simple. Just to shorten the definitions, we consider loop-free graphs. An undirected edge is handled as a pair of opposite directed edges. Each vertex has a label in a set C that we first take equal to $[k] := \{1, ..., k\}$ for some positive integer k. We denote by $\pi(G)$ the set of labels of the vertices of a graph G and by $\pi_G(x)$ the label of a vertex x. The operations on graphs are $\oplus$, the union of disjoint graphs, the unary relabelling $relab_h$ that changes every label a into $h(a)$ (where h is a mapping from C to C) and the unary edge-addition $\overrightarrow{add}_{a,b}$ that adds directed edges from every vertex labelled a to every vertex labelled b where $a \neq b$. Since we wish to define simple graphs, parallel edges are fused (hence $\overrightarrow{add}_{a,b}(\overrightarrow{add}_{a,b}(G)) = \overrightarrow{add}_{a,b}(G)$). A different interpretation of $\overrightarrow{add}_{a,b}$ can be given (cf. Section 5) so as to define graphs with multiple edges. For constructing undirected graphs, we use $\overrightarrow{add}_{a,b} \circ \overrightarrow{add}_{b,a}$ that we abbreviate into $add_{a,b}$. We will denote $relab_h$ by $relab_{a\to b}$ if $h(a) = b$ and $h(c) = c$ if $c \neq a$. The constant symbol $\mathbf{a}$ denotes one vertex (with no edge) labelled by $a \in C$. We let F_k be the set of these operations and constant symbols. Every term t in $T(F_k)$ is called a *k-expression* and defines a graph $G(t)$ with vertex set equal to the set occurences of the constant symbols in t. A graph has *clique-width* at most k if it is isomorphic to $G(t)$ for some t in $T(F_k)$.

Terms representing graphs and properties of their vertices

Let $P(X_1, ..., X_n)$ be a property of sets of vertices $X_1, ..., X_n$ of a graph $G(t)$, denoted by a term t in $T(F_k)$. Here are some examples of properties: $Link(X, Y)$: there is an edge from some x in X to some y in Y; $Dom(X, Y)$: for every x in X, there is an edge from some y in Y to x; $Path(X, Y)$: X has two vertices linked by a path in $und(G[Y])$ ($G[Y]$ is

the subgraph of G induced on Y and $und(G[Y])$ is the corresponding undirected graph) and $Conn(X) : G[X]$ is connected.

We let $F_k^{(n)}$ be obtained from F_k by replacing each constant $\mathbf{a}$ by the constants $(\mathbf{a}, w)$ where $w \in \{0,1\}^n$ and we let $pr : F_k^{(n)} \to F_k$ be the mapping that erases the sequences w. It extends into $pr : T(F_k^{(n)}) \to T(F_k)$. A term t in $T(F_k^{(n)})$ defines the graph $G(pr(t))$ and the n-tuple of sets of vertices $(A_1, ..., A_n)$ such that A_i is the set of vertices which are occurrences of constant symbols $(\mathbf{a}, w)$ such that the i-th component of w is 1. Then, if $P(X_1, ..., X_n)$ is a property as above, we define $L_{P(X_1,...,X_n),k}$ as the set of terms t in $T(F_k^{(n)})$ such that $P(A_1, ..., A_n)$ is true in $G(pr(t))$, where $(A_1, ..., A_n)$ is the n-tuple of sets of vertices encoded by t.

3 Monadic second-order logic

Graph properties can be expressed by monadic second-order formulas (more generally by formulas of any logical language) via two (main) representations of graphs by relational structures. The first representation associates with every graph G the logical structure $\lfloor G \rfloor := \langle V_G, edg_G \rangle$ where edg_G is the binary relation on vertices such that $(x, y) \in edg_G$ if and only if there is an edge from x to y; the relation edg_G is symmetric if G is undirected. The second representation will be discussed below in Section 4.

Monadic second-order formulas will be written with the set variables $X_1, ..., X_n, ...$ (without first-order variables), with the atomic formulas $X_i \subseteq X_j$, $X_i = \emptyset$, $Sgl(X_i)$ (to mean that X_i denotes a singleton set) and $edg(X_i, X_j)$ (to mean that X_i and X_j denote singleton sets $\{x\}$ and $\{y\}$ such that $(x, y) \in edg_G$), and without universal quantifications. These synctactical constraints are not a loss of generality. A graph property $P(X_1, ..., X_n)$, where $X_1, ..., X_n$ denote sets of vertices, is an *MS graph property* if there exists an MS formula $\varphi(X_1, ..., X_n)$ such that, for every graph G and for all sets of vertices $X_1, ..., X_n$ of this graph, we have:

$$\lfloor G \rfloor \models \varphi(X_1, ..., X_n) \text{ if and only if } P(X_1, ..., X_n) \text{ is true in } G.$$

For each MS property $P(X_1, ..., X_n)$, the set of terms $L_{P(X_1,...,X_n),k}$ is regular, hence is the set accepted by a finite automaton over the functional signature $F_k^{(n)}$. However, the corresponding automata are frequently much too large to be constructed. This is due partly to the level of nesting of negations in the formulas but also to the number k: for example, the number of states of the minimal automaton recognizing $L_{Conn(X_1),k}$ is a two-level exponential in k. Instead of trying to construct automata for the most general sentences, we will restrict our attention to particular but expressive ones (and we will address later the difficulty concerning k).

Definition 1: $\exists MS(\mathcal{P})$ **sentences**

We let $\mathcal{P}$ be a set of MS graph properties consisting of the properties defined by the atomic formulas and of basic properties such as $Link(X_1, X_2)$, $Path(X_1, X_2)$, $Conn(X_1)$. We let $\{X_1, ..., X_n\}$ be a set of set variables. A *Boolean set term* is a term written with these variables, the operations $\cap$, $\cup$ and complementation. For example, $S = X_1 \cup \overline{X_3}$. A $\mathcal{P}$*-atomic formula* is a formula of the form $P(S_1, ..., S_m)$ where $S_1, ..., S_m$ are Boolean set terms and P belongs to $\mathcal{P}$. An $\exists MS(\mathcal{P})$ sentence is a sentence of the form $\exists X_1, ..., X_n.\varphi$ where φ is a positive Boolean combination of $\mathcal{P}$-atomic formulas. Note that this definition depends on a set $\mathcal{P}$ that we leave "extensible", according to the needs.

Examples 2: We now give some examples of properties of simple undirected graphs expressible by $\exists MS(\mathcal{P})$ sentences.

(1) The property of *p-vertex colorability* is expressed by the sentence :

$$\exists X_1, ..., X_p \ .(Part(X_1, ..., X_p) \wedge St(X_1) \wedge ... \wedge St(X_p))$$

where $Part(X_1, ..., X_p)$ expresses that $X_1, ..., X_p$ define a partition of the vertex set and $St(X_i)$ expresses that X_i is *stable*, i.e., that the induced graph $G[X_i]$ has no edge. A p-vertex coloring defined by $X_1, ..., X_p$ is *acyclic* if furthermore, each induced graph $G[X_i \cup X_j]$ is acyclic (i.e., is a forest). The existence of an acyclic p-coloring for G (we will say that G is *p-AC-colorable*) is expressed by:

$$\begin{array}{c}\exists X_1, ..., X_p \ .(Part(X_1, ..., X_p) \wedge St(X_1) \wedge \\ ... \wedge St(X_p) \wedge ... \wedge NoCycle(X_i \cup X_j) \wedge ...)\end{array}$$

with one formula $NoCycle(X_i \cup X_j)$ for all i, j with $1 \leq i < j \leq p$.

(2) *Minor inclusion.* Let H be a simple, loop-free and undirected graph with vertex set $\{v_1, ..., v_p\}$. A graph G contains H as a minor if and only if it satisfies the sentence :

$$\begin{array}{c}\exists X_1, ..., X_p \ .(Disjoint(X_1, ..., X_p) \wedge Conn(X_1) \wedge ... \\ \wedge Conn(X_p) \wedge ... \wedge Link(X_i, X_j) \wedge ...)\end{array}$$

where $Disjoint(X_1, ..., X_p)$ expresses that $X_1, ..., X_p$ are pairwise disjoint; there is one formula $Link(X_i, X_j)$ for every edge of H that links v_i and v_j.

(3) *Perfect graphs.* A (simple, loop-free and undirected) graph G is *perfect* if the chromatic number of each induced subgraph H is equal to the maximum size of a clique in H. This definition is not monadic second-order expressible (because the fact that two sets have equal cardinalities is not) but the characterization established by Chudnovsky et al. [3] in terms of excluded holes and antiholes is. A *hole* is an induced cycle of odd length at least 5 and an *antihole* is the edge-complement of a hole. A graph has a hole if and only if it satisfies the following sentence:

$$\begin{array}{c}\exists X, Y, Z, U, V.(Disjoint(X, Y, Z, U, V) \wedge edg(Z, U) \wedge edg(U, V) \wedge \\ \neg edg(Z, V) \wedge \deg_2(X, Z \cup Y) \wedge \deg_0(X, U \cup V) \wedge \\ \deg_2(Y, X \cup V) \wedge \deg_0(Y, U \cup Z) \wedge \deg_2(V, U \cup Y) \wedge \deg_2(Z, U \cup X))\end{array}$$

where $\deg_0(X, Y)$ means that $X \cap Y = \emptyset$, X is stable and not empty and there is no edge between X and Y; the property $\deg_2(X, Y)$ means that $X \cap Y = \emptyset$, X is stable and not empty and every vertex in X has exactly 2 neighbours in Y. For every term $t \in T(F_k)$, one can construct (easily) a term $\overline{t} \in T(F_{2k})$ that defines the edge complement of the graph $G(t)$ ([10]). We obtain that $G(t)$ is perfect if and only if the F_{2k}-automaton for holes rejects both t and $\overline{t}$. The algorithm of [2] can test if a graph is perfect in time $O(n^9)$ (n is the number of vertices). From the above logical expression of holes, we get a fixed-parameter *cubic* algorithm for testing perfectness (with clique-width or even tree-width as parameter).

(4) *Constrained domination* and other problems. Let $P(X_1) \in \mathcal{P}$. The sentence $\exists X.(P(X) \wedge Dom(\overline{X}, X))$ expresses that there exists a set X satisfying property P that

dominates all other vertices. Many vertex *partitionning problems* considered in [23] can be expressed by $\exists MS(\mathcal{P})$ sentences in similar ways.

From $\exists MS(\mathcal{P})$ sentences to automata

We review the main steps of the inductive construction of an automaton associated with a sentence of $\exists MS(\mathcal{P})$. (See [4] for automata on terms). We first consider the $\mathcal{P}$-atomic formulas. We assume that for each property $P(X_1, ..., X_m)$ of $\mathcal{P}$ and each k, we have constructed a finite automaton $\mathcal{A}_{P(X_1,...,X_m),k}$ that accepts the set of terms $L_{P(X_1,...,X_m),k}$. Actually, these automata depend on k in a uniform way (see the end of this section for the use of this observation).

Claim 3 : For set terms $S_1, ..., S_m$ over $\{X_1, ..., X_n\}$, the set of terms $L_{P(S_1,...,S_m),(X_1,...,X_n),k}$ is $h^{-1}(L_{P(X_1,...,X_m),k})$ where h is an *alphabetic homomorphism*: $T(F_k^{(n)}) \to T(F_k^{(m)})$ that replaces each constant symbol $(\mathbf{a}, w)$ for $w \in \{0,1\}^n$ by $(\mathbf{a}, w')$ for some $w' \in \{0,1\}^m$ and does not modify the nonnullary function symbols.

We only give an example: consider a property $P(X_1)$, $n = 3$ and $S = X_1 \cup \overline{X_3}$. Then $L_{P(S),(X_1,X_2,X_3),k} = h^{-1}(L_{P(X_1),k})$ where, for every $x = 0, 1$:

$$h(1x0) = h(1x1) = h(0x0) = 1 \text{ and } h(0x1) = 0,$$

i.e., $h(x_1, x_2, x_3) = x_1 \vee \neg x_3$, hence h encodes the set term S in a natural way. The subscript (X_1, X_2, X_3) in $L_{P(S),(X_1,X_2,X_3),k}$ indicates that, although $P(S)$ depends only on X_1 and X_3, the set of terms is defined as if $P(S)$ depended on X_1, X_2 and X_3.

From an automaton $\mathcal{A}_{P(X_1,...,X_m),k}$ that accepts $L_{P(X_1,...,X_m),k}$ one gets an automaton $\mathcal{A}_{P(S_1,...,S_m),(X_1,...,X_n),k}$ with same number of states (but more transitions in many cases) that accepts $L_{P(S_1,...,S_m),(X_1,...,X_n),k}$. If $\mathcal{A}_{P(X_1,...,X_m),k}$ is deterministic, then the automaton $\mathcal{A}_{P(S_1,...,S_m),(X_1,...,X_n),k}$ is also deterministic. This claim can also be used if the terms $S_1, ..., S_m$ are just variables, say X_{i_1},..., X_{i_m}, hence for handling a substitution of variables.

Claim 4 : If φ is a positive Boolean combination of $\mathcal{P}$-atomic formulas $\alpha_1, ..., \alpha_d$ for which we have constructed complete non-deterministic (resp. deterministic) automata $\mathcal{A}_1, ..., \mathcal{A}_d$ with respectively $N_1, ..., N_d$ states, one can construct a complete *product* non-deterministic (resp. deterministic) automaton for φ with $N_1 \times ... \times N_d$ states (or less after deletion of useless states).

Claim 5 : If θ is the sentence $\exists X_1, ..., X_n.\varphi$, and we have constructed an automaton $\mathcal{A}$ recognizing $L_{\varphi(X_1,...,X_n),k}$, we can obtain one recognizing $L_{\theta,k}$, with the same number of states by applying the mapping pr that deletes the sequences of Booleans from the constant symbols of $F_k^{(n)}$. The automaton for θ is not deterministic in general, even if $\mathcal{A}$ is. If $\mathcal{A}$ is deterministic, this construction defines an automaton with 2^n transitions associated with each constant symbol $\mathbf{a}$. Hence, the values of n should not be too large.

By these claims, one can construct for every k and every sentence φ in $\exists MS(\mathcal{P})$ a nondeterministic automaton $\mathcal{A}_{\varphi,k}$ that accepts the regular set of terms $L_{P,k} = L_{\varphi,k}$ where P is the property expressed by φ, provided the automata for the atomic formulas and the properties of $\mathcal{P}$ are known.

Automata for the atomic and basic formulas.

We cannot detail all constructions, but we consider as an example the automaton $\mathcal{A} := \mathcal{A}_{edg(X_1,X_2),k}$. Its set of states is S consisting of $Ok, Error, 0, a(1),\ a(2), ab$ for all $a, b \in [k]$, $a \neq b$. It has $k^2 + k + 3$ states. Their meanings are described in Table 1. This table shows for each state s the property P_s that it encodes: s is the state reached by the automaton after reading a term t in $T(F_k^{(2)})$ if and only if P_s holds for this term. In its description, (V_1, V_2) is the pair of sets of vertices of the graph $G(pr(t))$ (that we denote more simply by $G(t)$) encoded by the Boolean components of the constants occurring in t. Table 2 specifies the transitions. All transitions not listed go to $Error$. Ok is the accepting state.

State s	Property P_s
0	$V_1 = V_2 = \emptyset$
$a(1)$	$V_1 = \{v\}, V_2 = \emptyset, \pi_{G(t)}(v) = a$
$a(2)$	$V_1 = \emptyset, V_2 = \{v\}, \pi_{G(t)}(v) = a$
Ok	$V_1 = \{v_1\}, V_2 = \{v_2\},\ (v_1, v_2) \in edg_{G(t)}$
ab	$V_1 = \{v_1\}, V_2 = \{v_2\},\ v_1 \neq v_2,\ \ \pi_{G(t)}(v_1) = a,$ $\pi_{G(t)}(v_2) = b$ and $(v_1, v_2) \notin edg_{G(t)}$
$Error$	All other cases

Table 1 Meanings of the states of $\mathcal{A}$.

Transition rules	Conditions
$(\mathbf{a}, 00) \to 0$ $(\mathbf{a}, 10) \to a(1)$ $(\mathbf{a}, 01) \to a(2)$ $(\mathbf{a}, 11) \to Error$	
$relab_h[s] \to s$ $relab_h[a(i)] \to h(a)(i)$ $relab_h[ab] \to cd$	$s \in \{0, Ok\}$ $i \in \{1, 2\}$ $c = h(a), d = h(b), c \neq d$
$\overrightarrow{add}_{a,b}[s] \to s$ $\overrightarrow{add}_{a,b}[ab] \to Ok$	$s \neq ab$
$\oplus[a(1), b(2)] \to ab$ $\oplus[b(2), a(1)] \to ab$ $\oplus[a(2), b(1)] \to ba$ $\oplus[b(1), a(2)] \to ba$ $\oplus[s, 0] \to s$ $\oplus[0, s] \to s$	$a \neq b$ $s \in S$

Table 2 The transition rules of $\mathcal{A}$.

We have one automaton for each k, but all these automata have a same concise description. (If we let C be the set of positive integers, we can consider that Table 2 specifies a unique automaton with infinitely many states. We will use this observation when discussing below *fly-automata*.)

Theorem 6 : Let $\mathcal{P}$ be a set of basic graph properties. For each $P(Y_1, ..., Y_m) \in \mathcal{P}$ and for each k, let a deterministic automaton $\mathcal{A}_{P(Y_1,...,Y_m),k}$ with $N(P, k)$ states be already

specified. For every sentence θ of the form $\exists X_1, ..., X_n.\varphi$ where φ is a Boolean combination of $\mathcal{P}$-atomic formulas $\alpha_1, ..., \alpha_d$, a nondeterministic automaton $\mathcal{A}_{\theta,k}$ (over the signature F_k because θ has no free variables) having at most $N := N_1 \times ... \times N_d$ states can be constructed where $N_i := N(P_i, k)$ and P_i is the property used to define α_i. $\square$

A complete and deterministic automaton over F_k (where we use only the elementary relabellings $relab_{a\to b}$) with N states, has $k + 2k(k-1).N + N^2$ transitions. The automaton $\mathcal{A}_{\theta,k}$ has thus $k.2^n + 2k(k-1).N + N^2$ transitions. Its nondeterministic transitions are only associated with the constant symbols. In the following theorem, we let m be an upper-bound to the time necessary to determine the output state of a deterministic transition or the i-th output state of a nondeterministic one. With these hypotheses and notation:

Theorem 7 : For every term t in $T(F_k)$, one can decide in time $m.(2^n + N^2).\mid t \mid$ if the graph $G(t)$ satisfies θ.

Proof : We use a bottom-up computation on t to determine at each node u of its syntactic tree the set of states that can occur at u. For the q occurrences of constant symbols, this takes total time at most $m.2^n.q$. For the occurrences of unary and binary symbols, this takes total time at most $m.N^2.(\mid t \mid -q).\square$

Note that we do not determinize the automaton $\mathcal{A}_{\theta,k}$. We only compute the transitions of $det(\mathcal{A}_{\theta,k})$, the determinized automaton of $\mathcal{A}_{\theta,k}$, that are needed for checking a given term.

Some basic graph properties and their automata.

We classify the atomic formulas and some "basic" graph properties (to be included in $\mathcal{P}$) in terms of the numbers of states $N(k)$ of deterministic F_k-automata that check them.

Polynomial-sized automata: The automata for $X_1 \subseteq X_2$, $X_1 = \emptyset$, $Part(X_1, ..., X_p)$ and $Disjoint(X_1, ..., X_p)$ have 2 states, the one for $Sgl(X_1)$ has 3 states. For $edg(X_1, X_2)$, we have defined above an F_k-automaton with $k^2 + k + 3$ states. The F_k-automaton for the property that X_1 has at most p elements has $p + 2$ states.

Single-exponential sized automata: The F_k-automaton for $St(X_1)$ has $2^k + 1$ states. Those for $Link(X_1, X_2)$ and $Dom(X_1, X_2)$ have $2^{2k} + 1$ states.

For $Path(X_1, X_2)$, we can constructed a (non-minimal) F_k-automaton with less than 2^{k^2+2} states. For the property *maximum degree at most p,* we can construct an F_k-automaton with $2^{k^2 p \log(p)}$ states.

Double-exponential sized automata: For connectedness, we can build an F_k-automata with about 2^{2^k} states and the unique minimal F_k-automaton has more than $2^{2^{k/2}}$ states. However, if we have a upperbound p to the degree of the graphs to be checked, then an F_k-automaton with $2^{p.k^2}$ states suffices. For the property of being a forest, we can construct an F_k-automaton with $2^{2^{O(k)}}$ states but we have no lower bound showing that a double exponential is necessary.

Fly-automata

A *fly-automaton* is an automaton (possibly not deterministic) whose transition rules are not listed in a table but are defined by finitely many clauses that we can call *transition meta-rules*. Table 2 shows such rules. It shows actually the meta-rules of a fly-automaton

with infinitely many states (where we replace $[k]$ by the set of positive integers). Each time a transition is needed it is computed from the specifications of Table 2. On input t in $T(F_k)$, this infinite automaton only uses the rules concerning the states of the set S defined above (see Table 1).

The finite F_k-automata described by Table 2 have $O(k^4)$ transitions, which makes difficult to compile their rules in a table unless k is small. This is even impossible for automata like the ones for connectedness that have $2^{2^{\Theta(k)}}$ states. Their tables cannot be constructed, even for small values of k. Hence, using fly-automata is necessary in such cases.

The automata currently used in compilation are "small" (typically, they have to recognize the key words of a programming language) whereas the input words (programs) are much larger. In the present case, the situation is the opposite: the automata are huge and the input terms are "small" (typically, terms of size 200 to define graphs with 50 vertices). But since these automata have concise descriptions, instead of trying to *compile* them, we propose to *interpret* them, that is to compute only their transitions that are needed for particular input terms.

It is clear that the constructions of Claims 3 and 4 can be used for fly-automata. For example, the meta-rules for two automata $\mathcal{A}$ and $\mathcal{B}$ can be combined to form those of their product. The algorithm of Theorem 7 that checks if a term is accepted by a nondeterministic automaton without determinizing it is applicable to a nondeterministic fly-automaton such that finitely many transitions are possible at each node.

Experiments have been conducted by I. Durand with her software Autowrite that implements automata on terms [13]. Here are some results concerning colorability and acyclic colorability. Grünbaum has given an example of a 3-colorable planar graph with 6 vertices (the clique K_6 minus the 3 edges of a perfect matching) that is not 4-AC-colorable but is 5-AC-colorable. These facts have been verified in a few seconds by using a term in $T(F_3)$ of size 15 that defines this graph and in 94 minutes by using a term in $T(F_5)$ of size 21. The Petersen graph (10 vertices, 15 edges) is 3-colorable, not 3-AC-colorable, but it is 4-AC-colorable. This last fact has been verified in 17 minutes on a term in $T(F_7)$ that defines this graph. The corresponding automata are too large to be constructible.

4 Edge set quantifications

We will now consider graphs that can have multiple edges. Because of the chosen representation of graphs, the *MS* properties cannot take into account the multiplicity of edges: that a pair of vertices (x, y) belongs to edg_G does not tell us the exact number of edges from x to y. We define another representation to remedy this drawback. The *incidence graph* of an undirected graph G is the simple directed bipartite graph $Inc(G) := \langle V_G \cup E_G, in_G \rangle$ where in_G is the set of pairs (e, x) such that e belongs to the set of edges E_G and x is an end vertex of e. We no longer use the convention that an undirected edge is a pair of opposite directed edges, and we use the simpler notation in_G instead of $edg_{Inc(G)}$. If G is directed, we define $Inc(G) := \langle V_G \cup E_G, in_{1G}, in_{2G} \rangle$ where in_{1G} (resp. in_{2G}) is the set of pairs (e, x) such that $e \in E_G$ and x is the tail vertex of e (resp. its head vertex). Hence, $Inc(G)$ is directed and bipartite with two types of edges. We will denote by $\lceil G \rceil$ the graph $Inc(G)$ considered as a relational structure, either over $\{in\}$ or over $\{in_1, in_2\}$.

A graph property $P(X_1, ..., X_n, Y_1, ..., Y_m)$, where $X_1, ..., X_n$ denote sets of vertices and $Y_1, ..., Y_m$ denote sets of edges, is an *MS_2 graph property* if there exists an *MS* formula

$\varphi(X_1, ..., X_n, Y_1, ..., Y_m)$, such that, for every graph G, for all sets of vertices $X_1, ..., X_n$ and for all sets of edges $Y_1, ..., Y_m$, we have:

$$\lceil G \rceil \models \varphi(X_1, ..., X_n, Y_1, ..., Y_m)$$
$$\text{if and only if } \; P(X_1, ..., X_n, Y_1, ..., Y_m) \text{ is true in } G.$$

The property that a simple undirected graph has at least 3 vertices and a Hamiltonian cycle is an MS_2-property that is not MS (see [6], Chapter 5). Hence, using $\lceil G \rceil$ instead of $\lfloor G \rfloor$ improves, also for simple graphs, the expressive power of monadic second-order logic. The appropriate parameter in the FPT algorithms that check MS_2 graph properties is not clique-width but tree-width.

For each k, there is a finite set H_k of graph operations such that a graph has tree-width at most k if and only if it is defined by a term over H_k. Linear time algorithms can convert tree-decompositions (the well-known definition is recalled in the next section) into terms and vice-versa. For every *MS_2 graph property* expressed by an MS sentence φ and every integer k, one can construct a finite automaton that recognizes the set of terms in $T(H_k)$ that define graphs G such that $\lceil G \rceil \models \varphi$, i.e., that satisfy that property. However, since H_k uses the operation of *parallel-composition* (denoted by $//$) that combines graphs by fusing vertices instead of the disjoint union $\oplus$, the corresponding automata are much more complicated than those constructed above. This observation motivates the introduction of a special type of tree-decomposition, hence of a variant of tree-width, that lacking of a better term, we call *special tree-width*. The algebraic representation of the corresponding *special tree-decompositions* need not use parallel-composition.

5 Special tree-width

In order to simplify the presentation, we will only consider undirected graphs. However, the definitions and results extend easily to directed graphs. Our definition is based on the operations that define clique-width. We will use the operations $add_{a,b}$ instead of the operations $\overrightarrow{add}_{a,b}$. In order to define *graphs with multiple edges*, we will change the interpretation of the operation $add_{a,b}$ in the following way: if in a graph G there is already an edge between a vertex x labelled by a and a vertex y labelled by b, then the operation $add_{a,b}$ applied to G adds another edge between x and y. The corresponding notion of clique-width for graphs with multiple edges is studied in [7]. However, we will use here this feature in a restricted situation.

Definition 8: *Special terms*

We will use the graph operations that define clique-width (cf. Section 2). The labels of vertices will be taken from the sets $[k]_\perp := [k] \cup \{\perp\}$ instead of $[k]$ and the corresponding sets of operations will be denoted by $F_{k,\perp}$. (The label $\perp$ will be used as a default label.) We (still) denote by $\pi(G)$ the set of labels of the vertices of a graph G and by $\pi_1(G)$ the subset of those that label a single vertex of G. If $t \in T(F_{k,\perp})$, then $\pi(t)$ denotes $\pi(G(t))$ and $\pi_1(t)$ denotes $\pi_1(G(t))$.

A term t in $T(F_{k,\perp})$ is a *special term* if it satisfies the following conditions:

1) $\pi(t') - \pi_1(t') \subseteq \{\perp\}$ for every subterm t' of t (we consider t as one of its subterms),
2) if $t_1 \oplus t_2$ is a subterm of t, then $\pi(t_1) \cap \pi(t_2) \subseteq \{\perp\}$,
3) for every relabelling $relab_h$ occurring in t, we have $h(\perp) = \perp$,
4) for every operation $add_{a,b}$ that occurs in t, we have $a \neq \perp$ and $b \neq \perp$,
5) the constant symbol $\perp$ has no occurrence in t.

We denote by $SpT(F_{k,\perp})$ the sets of special terms in $T(F_{k,\perp})$. The *special tree-width* of a graph G, denoted by $sptwd(G)$, is the least integer k such that $G = G(t)$ for some term t in $SpT(F_{k+1,\perp})$. The comparison with tree-width will justify the "+1" in the definition. The special tree-width of a graph consisting of isolated vertices is 0. Since the sets $\pi(t)$ and $\pi_1(t)$ are computable inductively on the structure of a term t, the sets $SpT(F_{k+1,\perp})$ are regular.

A graph defined by a special term in $SpT(F_{k,\perp})$ has at most one vertex labelled by each a in $[k]$ and possibly several vertices labelled by $\perp$. No new edge can be added between vertices such that one of them is labelled by $\perp$. These vertices are somehow "terminated". Furthermore, each occurrence of an operation $add_{a,b}$ adds at most one edge (by Conditions 1) and 4)). It may add no edge if the argument graph has no vertex labelled by a or no vertex labelled by b. We will say that such an occurrence is *useful* if it adds an edge. (Occurrences that are not useful can be deleted, which gives a smaller *reduced* term defining the same graph.) The graph $G(t)$ defined by a special term t can be constructed with vertex set $Occ_0(t)$, the set of occurrences of constant symbols in t, and edge set $Occ_1(t)$, the set of useful occurrences of edge addition operations. This remark will be used in Section 6.

Example: Trees have special tree-width 1. An undirected tree with one distinguished node called its *root*, is labelled as follows: the root is labelled by 1, all other nodes by $\perp$. Let T_1, T_2 be two such trees, defined by terms $t_1, t_2 \in SpT(F_{2,\perp})$. Then, we let $T := T_1 \ltimes T_2$ be defined by the term

$$t := relab_{2\longrightarrow\perp}\ (add_{1,2}(t_1 \oplus relab_{1\longrightarrow 2}(t_2))) \in SpT(F_{2,\perp}).$$

This tree is built as the disjoint union of the trees T_1 and T_2 augmented with an undirected edge between their roots, and the root of T is defined as that of T_1. Every rooted and undirected tree is generated by $\ltimes$ from the trees reduced to isolated roots, that are defined (up to isomorphism) by the constant symbol $\mathbf{1}$. Hence, every rooted and undirected tree is defined by a term in $SpT(F_{2,\perp})$. One can forget the root by applying the operation $relab_{1\longrightarrow\perp}$.□

We now consider tree-decompositions. A rooted and directed tree T is directed from the root towards the leaves.

Definition 9: A *special tree-decomposition* of a graph G is a pair (T, f) such that T is a rooted and directed tree with set of nodes N_T and $f : N_T \longrightarrow \mathcal{P}(V_G)$ is a mapping such that:
1) Every vertex of G belongs to $f(u)$ for some u in N_T.
2) Every edge has its ends in $f(u)$ for some u in N_T.
3) For each vertex x, the set $f^{-1}(x) := \{u \in N_T \mid x \in f(u)\}$ is a directed path in T.

Condition 3) characterizes special tree-decompositions. The *width* of a decomposition (T, f) is the the maximal cardinality minus 1 of a *box*, i.e. of a set $f(u)$. A *path-decomposition* is defined as a tree-decomposition such that T is a directed path (hence it is special). The *tree-width* $twd(G)$ (the *path-width* $pwd(G)$) of a graph G is the minimal width of a tree-decomposition (a path-decomposition) of this graph. It is known from [5,10] that a set of simple graphs, directed or not, that has bounded tree-width has bounded clique-width: if G undirected has tree-width k, then it has clique-width at most 3.2^{k-1} and in some cases, more that $2^{k/2}$. However, its clique-width is at most $pwd(G) + 2$.

Proposition 10: The special tree-width of a graph is the minimal width of a special tree-decomposition of this graph. There are linear-time algorithms for converting a term t in $SpT(F_{k+1,\perp})$ into a special tree-decomposition of width k of the graph $G(t)$ and vice-versa.

Proposition 11: For every graph G we have:
(1) $twd(G) \leq sptwd(G) \leq pwd(G)$,
(2) $cwd(G) \leq sptwd(G) + 2$.

These facts are clear from Proposition 10 and the definitions. Note that clique-width behaves with respect to special tree-width exactly as with respect to path-width, and without the exponential increase. We will denote by $STWD(\leq k)$ the class of undirected graphs of special tree-width at most k.

Proposition 12: For each k, the class $STWD(\leq k)$ is closed under the following transformations:
1) Removal of vertices and edges,
2) addition of edges parallel to existing edges,
3) smoothing vertices of degree 2. □

Smoothing a vertex of degree 2 means contracting any one of its two incident edges. For the case of directed graphs (see [7]), reversals of edge directions also preserve special tree-width, whereas they do not preserve clique-width. It follows from items 1) and 3) of Proposition 12 that the class $STWD(\leq k)$ is closed under taking *topological minors* ([11]). It is not closed under taking minors as we will see in Proposition 16. In the following proposition, $pwd(L)$ denotes the least upper bound of the path-widths of the graphs in a set L and similarly for the other notions of width.

Proposition 13: The class of graphs of tree-width 2 has unbounded special tree-width. For every set of graphs L:

$$pwd(L) < \infty \Longrightarrow sptwd(L) < \infty \Longrightarrow twd(L) < \infty \text{ and}$$

$$sptwd(L) < \infty \Longrightarrow cwd(L) < \infty,$$

whereas the converse implications do not hold.□

Proof: We will use the following claim (where $G \otimes *$ is G augmented with a new vertex $*$ and edges between it and all vertices of G):

For every graph G, the special tree-width of $G \otimes *$ is equal to its path-width.

For proving the first assertion, we assume that every graph of tree-width 2 has special tree-width at most k. If T is any tree, then $T \otimes *$ has tree-width at most 2, hence special tree-width at most k, and path-width at most k by the claim. It follows that T, since it is a subgraph of $T \otimes *$, has path-width at most k, but trees have unbounded path-width ([11]), which gives a contradiction.

The implications follow from Proposition 11. Trees have special tree-width at most 1 and unbounded path-width. Graphs of tree-width 2 have unbounded special tree-width, hence the opposite implications are false. The converse of $sptwd(L) < \infty \Longrightarrow cwd(L) < \infty$ is false if L the set of cliques because it is of maximal clique-width 2 and of unbounded tree-width and special tree-width. □

Definition 14: A *tree-partition* of a graph G is a pair (T, f) such that T is a rooted tree with set of nodes N_T and $f : N_T \longrightarrow \mathcal{P}(V_G)$ is a mapping such that:

1) Every vertex of G belongs to $f(u)$ for a unique node u of T,
2) Every edge has its two ends in some box or in two boxes $f(u)$ and $f(v)$ such that v is the father of u.

The *width* of (T, f) is defined as the maximal cardinality of a box, (no -1 here !), and the *tree-partition-width* (also called *strong tree-width*) of a graph G is the minimal width of its tree-partitions. We denote it by $tpwd(G)$. The *wheels*, i.e., the graphs $C_n \otimes *$ where C_n is the undirected cycle with n vertices ($n \geq 3$) have path-width (and special tree-width) 3 but unbounded tree-partition width (see [1, 24]). $MaxDeg(G)$ denotes the maximum degree of a graph G. The proof of the following proposition uses results from [24].

Proposition 15: For every graph G :

1) $sptwd(G) \leq 2.tpwd(G) - 1$,
2) $sptwd(G) \leq 20.(twd(G) + 1).MaxDeg(G)$.

A set of graphs of bounded degree has bounded special-tree-width if and only if it has bounded tree-width.□

This result suggests a question:

Which conditions on a set of graphs, other than bounded degree, imply that it has bounded tree-width if and only if it has bounded special tree-width?

Planarity does not since the graphs of tree-width at most 2 are planar but of unbounded special tree-width. From this case, we can see that conditions like excluding a fixed graph as minor or being uniformly k-sparse for some k do not either. All these conditions however, imply that, for simple graphs, bounded tree-width is equivalent to bounded clique-width (see [6], Chapter 2).

Proposition 16: Every graph of tree-width k is obtained by edge contractions from a graph of special tree-width at most $2k + 1$. The class of graphs of special tree-width at most k is not closed under taking minors for any $k \geq 5$.

Proposition 17: The special tree-width of a graph is the maximal special tree-width of its connected components. It is at most one plus the maximal special tree-width of its biconnected components. This upper bound is tight.

Open question 18: *The parsing problem*: Does there exist fixed functions f and g and an approximation algorithm able to do the following in time $O(n^{g(k)})$, where n is the number of vertices of the given graph:

Given a simple graph G and an integer k, either it answers (correctly) that G has special tree-width more than k, or it outputs a special term witnessing that its special tree-width is at most $f(k)$?

Stronger requirements would be that $f(k) = k$, giving an exact algorithm and/or the computation time $O(g(k).n^c)$ for some fixed c instead of $O(n^{g(k)})$. Since by a result by Bodlaender (presented in detail in [12]) such an algorithm exists for tree-width, with $f(k) = k$ and $c = 1$, one can think that this algorithm can be adapted in order to find special tree-decompositions.

6 Automata for monadic second-order formulas with edge set quantifications

Our objective is to adapt the constructions of Section 3 to the model-checking of MS_2 graph properties for graphs defined by special terms. We will obtain fixed-parameter linear algorithms for graphs of bounded special tree-width given by the relevant terms or decompositions.

MS_2 formulas and the encoding of assignments

In order to use MS_2-formulas, i.e., monadic second-order formulas with edge set quantifications, we will represent a graph G by the relational structure $\lceil G \rceil := Inc(G)$ defined in Section 4. As in Section 3, we will use formulas written without first-order variables and universal quantifications. We will use the "standard" set variables $X_1, ..., X_n, ...$ for denoting sets of vertices and similarly the variables $Y_1, ..., Y_m, ...$ for denoting sets of edges. The atomic formulas are of the forms $edg(X_i, X_j)$, $in(Y_i, X_j)$ (graphs are undirected), and of course, $X_i \subseteq X_j$, $Y_i \subseteq Y_j$, $Z = \emptyset$, $Sgl(Z)$, where Z is X_i or Y_j. The meaning of $in(Y_i, X_j)$ is that Y_i and X_j are singletons, respectively $\{y\}$ and $\{x\}$ such that $(y, x) \in in_G$.

We now discuss the encoding of assignments in terms. Let t be a special term and $G(t)$ be the (concrete) graph it defines. Its vertex set is $Occ_0(t)$, the set of occurrences of constant symbols and it edge set is $Occ_1(t)$, the set of useful occurrences of edge addition operations (cf. Definition 8.) In order to encode $\{X_1, ..., X_n, Y_1, ..., Y_m\}$-assignments, we will use, the signatures $F^{(n,m)}_{k,\perp}$ obtained from $F^{(n)}_{k,\perp}$ by replacing every edge addition operation f by the unary operations (f, w), for all w in $\{0, 1\}^m$.

We will use the projections pr as in Claim 5 and the projections pr', that delete the Booleans in the unary operations (f, w). It is clear that a term $t \in T(F^{(n,m)}_{k,\perp})$ such that $pr(pr'(t))$ is a special term and the occurrences of edge addition operations in $pr(pr'(t))$ are all useful, defines a graph $G(pr(pr'(t))$ and an $\{X_1, ..., X_n, Y_1, ..., Y_m\}$-assignment γ such that $\gamma(X_i)$ is a set of vertices (for $i \in [n]$) and $\gamma(Y_j)$ is a set of edges (for $j \in [m]$).

We will denote by $RT(F^{(n,m)}_{k,\perp}) \subseteq SpT(F^{(n,m)}_{k,\perp})$ the set of *reduced terms*, defined as the set of special terms in which every occurrence of an edge addition operation is useful. Whether a term t in $T(F^{(n,m)}_{k,\perp})$ is in $RT(F^{(n,m)}_{k,\perp})$ or not does not depend on the Boolean components of its constant symbols and of its edge addition operations. In other words, $RT(F^{(n,m)}_{k,\perp}) = pr'^{-1}(pr^{-1}(RT(F_{k,\perp})))$.

Claim 19 : For each triple n, m, k of integers, the set $RT(F^{(n,m)}_{k,\perp})$ is regular and is recognized by a deterministic automaton with 2^k states.

For every MS_2 formula φ with free variables in $\{X_1, ..., X_n, Y_1, ..., Y_m\}$ and every k, we define $L_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}$ as the set of terms t in $RT(F^{(n,m)}_{k,\perp})$ such that $(\lceil G(pr(pr'(t)))\rceil, \gamma(t)) \models \varphi$ where $\gamma(t)$ denotes the $\{X_1, ..., X_n, Y_1, ..., Y_m\}$-assignment encoded by t. The language

$L_{P(X_1,...,X_n,Y_1,...,Y_m),k}$ can be defined similarly for a graph property $P(X_1, ..., X_n, Y_1, ..., Y_m)$ independently of its logical expression. Note that we define here sets of reduced terms.

Theorem 20: For every MS_2 graph property $P(X_1, ..., X_n, Y_1, ..., Y_m)$ and every k, the language $L_{P(X_1,...,X_n,Y_1,...,Y_m),k}$ is regular and an automaton recognizing it can be constructed from an MS_2 formula that expresses P.

Proof: The construction is as for Theorem 6. At each step we restrict the defined sets so that they only contain reduced terms. For example, if φ is $\neg\theta$, we construct an automaton that recognizes $L_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k} = RT(F^{(n,m)}_{k,\perp}) - L_{\theta,(X_1,...,X_n,Y_1,...,Y_m),k}$. We do not simply take the complement.

Let us say a few words on the automata for the atomic formulas. Most of the constructions are straightforward from the definitions, as in Theorem 6. We only consider the atomic formulas $edg(X_1, X_2)$ and $in(Y_1, X_1)$.

An $F^{(2)}_{k,\perp}$-automaton $\mathcal{A}'$ for $edg(X_1, X_2)$, that is essentially the same as the automaton $\mathcal{A}$ of Theorem 6, can be constructed so as to work correctly on reduced terms. The automaton $\mathcal{A}_{edg(X_1,X_2),k}$ intended to define the set $L_{edg(X_1,X_2),k}$ is then obtained by a product with the one of Claim 19 that recognizes the set of reduced terms. Its number of states is thus $2^k.O(k^2)$ instead of $O(k^2)$. In the following remark, we discuss this difficulty.

We now construct an automaton $\mathcal{B}$ for $in(Y_1, X_1)$, intended to work on reduced terms. Its set of states is $S := \{0, Error, Ok\} \cup [k]$. Their meanings are described in Table 3, where W_1 denotes the value of the set variable Y_1. Its transitions not yielding $Error$ are in Table 4. As examples of transitions to $Error$ we have $\oplus[Ok, a] \to Error$ and $(add_{a,b}, 1)[Ok] \to Error$. The unique accepting state is Ok.

State s	Property P_s
0	$V_1 = W_1 = \emptyset$
a	$V_1 = \{v\}, W_1 = \emptyset, \pi_{G(t)}(v) = a$
Ok	$V_1 = \{v\}$, $W_1 = \{e\}$, $(e, v) \in in_{G(t)}$
$Error$	All other cases

Table 3 Meanings of the states of $\mathcal{B}$.

Transition rules	Conditions
$(\mathbf{a}, 0) \to 0$ $(\mathbf{a}, 1) \to a$	
$relab_h[0] \to 0$ $relab_h[Ok] \to Ok$ $relab_h[a] \to b$	 $b = h(a) \neq \perp$
$(add_{a,b}, 0)[s] \to s$ $(add_{a,b}, 1)[c] \to Ok$	all s $c \in \{a, b\}$
$\oplus[s, 0] \to s$ $\oplus[0, s] \to s$	all s

Table 4 The transition rules of $\mathcal{B}$.

Remark 21: The above construction associates with each subformula $\theta(X_1, ..., X_n, Y_1, ..., Y_m)$ of the considered formula φ an automaton $\mathcal{A}_{\theta,(X_1,...,X_n,Y_1,...,Y_m),k}$ that recognizes only

reduced terms. This means that each of these automata repeats the verification that the input term is reduced. One can actually postpone this verification to the very end.

Assume that for each atomic formula $\alpha(X_1, ..., X_n, Y_1, ..., Y_m)$, we have an automaton $\mathcal{B}_{\alpha,(X_1,...,X_n,Y_1,...,Y_m),k}$ such that

$$L_{\alpha,(X_1,...,X_n,Y_1,...,Y_m),k} = L(\mathcal{B}_{\alpha,(X_1,...,X_n,Y_1,...,Y_m),k}) \cap RT(F_{k,\perp}^{(n,m)}).$$

This means that $\mathcal{B}_{\alpha(X_1,...,X_n,Y_1,...,Y_m),k}$ is constructed so as to work correctly on reduced terms, and this is what we did above for $\mathcal{A}'$ and $\mathcal{B}$.

Let us build $\mathcal{B}_{\varphi(X_1,...,X_n,Y_1,...,Y_m),k}$ for all formulas φ by applying the general inductive construction described for Theorem 6 with, for the negation:

$$L(\mathcal{B}_{\neg\theta,(X_1,...,X_n,Y_1,...,Y_m),k}) = T(F_{k,\perp}^{(n,m)}) - L(\mathcal{B}_{\theta,(X_1,...,X_n,Y_1,...,Y_m),k}).$$

At the end, for the input formula $\varphi(X_1, ..., X_n, Y_1, ..., Y_m)$, we make the restriction to reduced terms by defining $\mathcal{A}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}$ in such a way that:

$$L(\mathcal{A}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}) = L(\mathcal{B}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}) \cap RT(F_{k,\perp}^{(n,m)}).$$

Hence, we use only at the end the restriction to reduced terms. We claim that $L(\mathcal{A}_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}) = L_{\varphi,(X_1,...,X_n,Y_1,...,Y_m),k}$. This is true by the hypotheses on the automata $\mathcal{B}_\alpha$ associated with the atomic formulas and by the following observations:

if L, M, R, T, L', R' and T' are sets such that $L, M, R \subseteq T$ and $L', R' \subseteq T'$, and pr is a mapping from T' to T such that $T' = pr^{-1}(T)$ and $R' = pr^{-1}(R)$ then, $(L \cap R) \cap (M \cap R) = (L \cap M) \cap R$, $(L \cap R) \cup (M \cap R) = (L \cup M) \cap R$, $R - (L \cap R) = (T - L) \cap R$ and $pr(L' \cap R') \cap R = pr(L') \cap R$.

□

$\exists MS_2(\mathcal{P})$ sentences can be defined and fly-automata can be used to check them on graphs of bounded special tree-width.

Tree-width versus special tree-width

We now explain why the constructions of automata are easier for bounded special tree-width than for bounded tree-width.

Definition 22: *Special tree-width-terms.*

We let $H_{k,\perp}$ be the signature obtained from $F_{k,\perp}$ by replacing the operation $\oplus$ by $//$. This operation symbol will be interpreted as follows: for graphs G and H such that, as in Definition 8, $\pi(G) - \pi_1(G) \subseteq \{\perp\}$ and $\pi(H) - \pi_1(H) \subseteq \{\perp\}$, we define $G//H$ from $G \oplus H$ by fusing any two vertices having the same label $a \neq \perp$. A *special tree-width-term* is a term t in $T(H_{k,\perp})$ that satisfies conditions 1), 3), 4) and 5) of Definition 8. We denote by $TWT(H_{k,\perp})$ the set of these terms. Every graph is the value $G(t)$ of a term t in $TWT(H_{k,\perp})$ for some large enough k.

Proposition 23 ([6], Chapter 2): The tree-width of a graph is the least integer k such that this graph is the value of a term in $TWT(H_{k+1,\perp})$. There are linear-time algorithms for converting a term t in $TWT(H_{k+1,\perp})$ into a tree-decomposition of width k of the graph $G(t)$ and vice-versa.

In view of the verification of MS_2 properties, let us consider the encoding of assignments in terms. Let $t \in TWT(H_{k,\perp})$ and $G = G(t)$. Its edges are in bijection with the set $Occ_1(t)$ defined as for special terms. However, its vertex set is isomorphic to a quotient of $Occ_0(t)$ by the equivalence relation $\approx$ expressing that x and y in $Occ_0(t)$ have a least common ancestor u that is an occurrence of // and that the associated vertices have the same label (different from $\perp$) in $G(t/u)$, the graph defined by the subterm of t issued from u. This means that x and y, because of a fusion occurring at u, yield the same vertex of G. Hence, we lose the nice bijection between vertices of $G(t)$ and particular occurrences of symbols in t. It follows that a set $X \subseteq Occ_0(t)$ represents correctly a set of vertices of $G(t)$ if and only it is saturated for $\approx$ (is a union of classes of this equivalence relation). The $H_{k,\perp}$-automaton analogous to $\mathcal{B}_{\varphi,(X_1,\ldots,X_n,Y_1,\ldots,Y_m),k}$ must check this saturation property, which increases substantially its number of states.

There is actually another possibility for representing vertices by occurrences of symbols in terms. Let us assume that all vertices of $G(t)$ are labelled by $\perp$ and that each vertex corresponds to a unique occurrence of an operation $relab_{a\longrightarrow\perp}$. Such occurrences, let us denote their set by $Occ_1^{vert}(t)$, can be chosen to represent the vertices. In this case, an edge will be represented by a node of t that is *below the nodes representing its ends*. This is not a difficulty for constructing an $H_{k,\perp}$-automaton for the atomic formula $in(Y_1, X_1)$ (like $\mathcal{B}$ in the proof of Theorem 20) having $k + 3$ states. However, the construction of an $H_{k,\perp}$-automata for $edg(X_1, X_2)$ is more complicated. It can be done directly or by the general construction: since $edg(X_1, X_2)$ is equivalent to $X_1 \neq X_2 \wedge \exists Y_1(in_1(Y_1, X_1) \wedge in_2(Y_1, X_2))$, the general construction produces an $H_{k,\perp}$-automaton with $2^{O(k^2)}$ states. (The factor k^2 is due to the use of a product of two automata with $k + 3$ states for the subformula $in_1(Y_1, X_1) \wedge in_2(Y_1, X_2)$, and the exponentiation is due to the determinization that is needed because of $\exists Y_1$). Furthermore, every deterministic $H_{k,\perp}$-automaton for $edg(X_1, X_2)$ *must have* at least $2^{k(k-1)}$ states ([6], Chapter 6). Hence, with this encoding of assignments, an atomic formula like $edg(X_1, X_2)$ needs already fairly "large" automata. This difficulty is avoided in special terms because they use $\oplus$ instead of //.

7 Conclusion

We have presented some tools intended to yield practically usable methods for the verification of *certain* monadic second-order graph properties for graphs of bounded tree-width or clique width. We have proposed to restrict the constructions of automata to the formulas of an appropriate fragment of monadic second-order logic and to use *fly automata* (a notion first presented in [8]). Although some experimental results are encouraging, these ideas have to be tested on more cases.

Special tree-width seems interesting on its own, but the construction of "small" automata has motivated its introduction. The corresponding parsing problem is open.

What about automata for graphs of bounded tree-width? We are presently working on a redundant representation of these graphs that equips terms in $TWT(H_{k,\perp})$ with additional labels. "Small" automata for $edg(X_1, X_2)$ whose transitions use these additional labels (as opposed to the operations of $H_{k,\perp}$) can then be constructed.

8 References

[1] H. Bodlaender, J. Engelfriet, Domino tree-width, *J. Algorithms* **24** (1997) 94-123.
[2] M. Chudnovsky *et al.*, Recognizing Berge graphs, *Combinatorica* **25** (2005) 143-186.

[3] M. Chudnovsky *et al.*, The strong perfect graph theorem, *Ann. Math.* **164** (2006) 51-229.
[4] H. Comon *et al.*, *Tree Automata Techniques and Applications*, On line for free at: http://tata.gforge.inria.fr/
[5] D. Corneil, U. Rotics, On the relationship between clique-width and tree-width. *SIAM J. Comput.* **34** (2005) 825-847.
[6] B. Courcelle, *Graph structure and monadic second-order logic*, book to be published by *Cambridge University Press*. Readable on: http://www.labri.fr/perso/courcell/Book/CourGGBook.pdf
[7] B. Courcelle, On the model-checking of monadic second-order formulas with edge set quantifications, May 2010, to appear in *Discrete Applied Mathematics*. Available from : http://hal.archives-ouvertes.fr/hal-00481735/fr/
[8] B. Courcelle, I. Durand, Verifying monadic second-order graph properties with tree automata, *3rd European Lisp Symposium*, May 2010, Lisbon, Informal proceedings edited by C. Rhodes, pp. 7-21. See: http://www.labri.fr/perso/courcell/ArticlesEnCours/BCDurandLISP.pdf
[9] B. Courcelle, J. Makowsky, U. Rotics, Linear time solvable optimization problems on graphs of bounded clique-width. *Theory of Computing Systems* **33** (2000) 125-150.
[10] B. Courcelle, S. Olariu, Upper bounds to the clique-width of graphs, *Discrete Applied Mathematics* **101** (2000) 77-114.
[11] R. Diestel, *Graph Theory*, 3rd edition, Springer, 2005.
[12] R. Downey, M. Fellows, *Parameterized complexity*, Springer-Verlag, 1999.
[13] I. Durand, Autowrite: A tool for term rewrite systems and tree automata, *Electronic Notes in Theoret. Comput. Sci.* **124** (2005) 29-49.
[14] J. Flum, M. Grohe, *Parametrized complexity theory*, Springer, 2006.
[15] M. Frick, M. Grohe: The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic* **130** (2004) 3-31
[16] R. Ganian, P. Hlineny, On parse trees and Myhill-Nerode-type tools for handling graphs of bounded rank-width. *Discrete Applied Mathematics* **158** (2010) 851-867
[17] R. Ganian, P. Hlineny, J.Obdrzalek, Better algorithms for satisfiability problems for formulas of bounded rank-width, 2010, arXiv:1006.5621v1[cs. DM]
[18] G. Gottlob, R. Pichler, F. Wei: Abduction with bounded tree-width: from theoretical tractability to practically efficient computation. Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, Chicago, AAAI Press, 2008, pp. 1541-1546
[19] G. Gottlob, R. Pichler, F. Wei: Monadic Datalog over finite structures with bounded tree-width, CoRR abs/0809.3140 (2008)
[20] P. Hlineny, S. Oum: Finding branch-decompositions and rank-decompo-sitions. *SIAM J. Comput.* **38** (2008) 1012-1032.
[21] L. Stockmeyer, A. Meyer, Cosmological lower bound on the circuit complexity of a small problem in logic. *J. ACM* **49** (2002) 753-784.
[22] M. Weyer, Decidability of S1S and S2S. in *Automata, Logics, and Infinite Games: A Guide to Current Research*. Lect. Notes Comp. Sci. **2500,** Springer, 2002, pp. 207-230.
[23] M. Rao, MSOL partitioning problems on graphs of bounded tree-width and clique-width. *Theor. Comput. Sci.* **377** (2007) 260-267.
[24] D. Wood, On tree-partition-width, *European J. Combin.* **30** (2009) 1245-1253.

On extracting computations from propositional proofs (a survey)

Pavel Pudlák

Institute of Mathematics, Academy of Sciences*
Prague, Czech Republic
pudlak@math.cas.cz

Abstract

This paper describes a project that aims at showing that propositional proofs of certain tautologies in weak proof system give upper bounds on the computational complexity of functions associated with the tautologies. Such bounds can potentially be used to prove (conditional or unconditional) lower bounds on the lengths of proofs of these tautologies and show separations of some weak proof systems. The prototype are the results showing the feasible interpolation property for resolution. In order to prove similar results for systems stronger than resolution one needs to define suitable generalizations of boolean circuits. We will survey the known results concerning this project and sketch in which direction we want to generalize them.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.30

1 Introduction

Proof complexity studies problems about formal systems that are related to similar problems in computational complexity. In particular, some complexity classes can be associated in a natural way with formal systems and one can translate problems about these classes to problems about the formal systems. The formal systems that we have in mind are weak arithmetical theories formalized in predicate logic and proof systems for propositional calculus. Our original reason for studying weak arithmetical theories was to show the unprovability of some open problems in computational complexity theory. Since the attempts to show that, say, $\mathbf{P} \neq \mathbf{NP}$ is unprovable in Peano Arithmetic completely failed, researchers focused on the study of much weaker theories. Unfortunately we are still unable to show such independence results even for the weakest theory in which polynomial time computations are formalizable. Nevertheless, a number of interesting results have been proven and new proof methods have been introduced.

In computational complexity the most important problems can be reduced to proving lower bounds on the circuit size of boolean functions. Similarly, in proof complexity one can reduce problems about unprovability in weak theories of arithmetic to proving lower bounds on the lengths of proofs of tautologies in certain proof systems. In the 1980s the pioneering work in the area of lower bounds on propositional proofs was done by Armin Haken, who proved exponential lower bounds on proofs in Resolution [6], and Miklos Ajtai, who proved superpolynomial lower bounds (later extended to exponential) on proofs in bounded depth Frege systems [1]. In the early 1990s Jan Krajíček introduced a new method for proving lower bounds on propositional proofs, which we now call *feasible interpolation* [8]. This enables one to reduce the task of proving lower bounds on the lengths of propositional proofs to the task of proving lower bounds on the circuit size of boolean functions defined from tautologies.

* also supported by the Insititue for Theoretical Computer Science (project 1M0545) and grant IAA100190902.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 30–41

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Although the problem of proving nontrivial lower bounds on the size of boolean circuits is a notoriously open problem, this reduction is very useful for the following two reasons. First, one can often show that the reduction gives *monotone* boolean circuits and then one can use the well-known exponential lower bounds on the size of such circuits. Secondly, even when the reduction to monotone boolean circuits is not possible and we only get general circuits, the reduction gives us important information that, for example, can be used to show lower bounds based on some conjectures from computational complexity theory.

The method of feasible interpolation has been successfully applied to obtain exponential lower bounds on the lengths of proofs in Resolution [9] and several other proof systems for propositional logic. However, this method fails for systems that are only a little stronger than resolution, unless some commonly accepted conjectures in computational complexity are false [13, 4, 3]. Therefore we started a project whose aim is to find structures that are more general than boolean circuits and prove for them generalized forms of feasible interpolation. Our motivation is not only to find new ways of proving lower bounds on the lengths of proofs, but also to study a question important per se: *can one extract computational information from any propositional proof?*

There are other approaches to the fundamental question of proving lower bounds on the lengths of propositional proofs, most notably the approach based on *proof complexity generators*, [10, 12]. Proof complexity generators are inspired by the concept of pseudorandom generators and the conjecture is that some pseudorandom generators can actually be used to construct tautologies with no polynomial proofs. However, if this is true than one must find connections between propositional proofs and computational complexity for strong proof systems. It is likely that before such connections are found for strong proof systems, they will be discovered for moderately strong systems. Therefore we focus on such propositional proof systems.

Another approach, also studied by Krajíček, is based on using a different *form* of tautologies that are used in the feasible interpolation theorems [11]. For such tautologies, it is conceivable that the associated computational problem is solvable in polynomial time even for stronger systems. In [11] such a connection is proved for a very special form of tautologies and proofs in bounded depth Frege systems.

Although our project has already yielded some preliminary results, it would be premature to try to describe them in this paper. We will rather focus on describing the results that we want to generalize, hoping that our presentation will be more understandable than the original ones in [17, 9].

2 Preliminaries

2.1 Propositional proof systems

The *Resolution* propositional proof system is a proof system for proving tautologies that are in DNF form. Given a tautology ϕ in DNF form, we take its negation, which is in CNF form, and treat it as a set of disjunctions, which are called *clauses*. A proof of ϕ in Resolution is a proof of contradiction from the clauses. In Resolution we treat clauses as sets of literals; a literal is a propositional variable or negated propositional variable. The single rule used in Resolution is:

$$\frac{\Gamma, p \qquad \Delta, \neg p}{\Gamma, \Delta}.$$

The contradiction, which is the last clause in the proof, is represented by the empty set.

A *Frege system* is any sound and complete propositional proof system that is based on a finite number of rules. No nontrivial lower bounds are known for Frege systems. A depth d Frege system is a Frege system in which only formulas of depth d are allowed. We want to present bounded depth Frege systems as generalizations of Resolution. Thus we will define a depth d Frege system as a *refutation* proof system based on sequents that use formulas of depth at most d. We will only use formulas with negations at variables; so $\neg\phi$ denotes the formula obtained from ϕ by switching conjunctions and disjunctions and switching literals. Defining the depth of a literal to be 1, we get that Resolution is the depth 1 Frege system.

For $d > 1$, the *cut rule* of Resolution is generalized to arbitrary formulas ϕ of depth at most d:

$$\frac{\Gamma, \phi \qquad \Delta, \neg\phi}{\Gamma, \Delta}.$$

Further, we also have *rules for introducing conjunctions and disjunctions:*

$$\frac{\Gamma, \phi \qquad \Delta, \bigwedge A}{\Gamma, \Delta, \phi \wedge \bigwedge A} \qquad \frac{\Gamma, \phi, \bigvee A}{\Gamma, \phi \vee \bigvee A},$$

and the *weakening rule:*

$$\frac{\Gamma}{\Gamma, \phi}.$$

We treat sequents as sets, and conjunctions and disjunctions as set operations in order not to have to introduce structural rules.

2.2 Polynomial search problems

Polynomial search problems, also called *Total Function Nondeterministic Polynomial search problems* and abbreviated by **TFNP,** are given by a binary relation R such that

1. $R(x, y)$ is decidable in deterministic polynomial time;
2. there exists a polynomial p such that for all x and y, if $R(x, y)$, then $|y| \leq p(|x|)$;
3. for every x there exists y such that $R(x, y)$.

Given such a relation and x, the task is to find y such that $R(x, y)$. There is a natural concept of polynomial reduction of one polynomial search problem to another one. Also many natural classes of polynomial search problems have been defined and they play an important role in proof complexity.

We will only mention one of these classes, *Polynomial Local Search,* or **PLS**. An instance of a Polynomial Local Search problem for a given input x is determined by a *search space* S, a *feasibility predicate* $F \subseteq S$, a *neighborhood function* $N : S \to S$, and a *cost function* $c : S \to \mathbb{N}$. The search space is $S = \{0, 1\}^m$, where m is polynomial in $|x|$. The functions N, c and the predicate F are computable in polynomial time. Formally, this means that the predicate F is parametrized by the input x and $F(s, x)$ is a binary relation in $\mathbf{P}$, etc. for the other notions. The functions and the predicate should satisfy:

1. $F(\bar{0})$;
2. if $F(s)$, then $F(N(s))$;
3. if $F(s)$ and $N(s) \neq s$, then $c(s) < c(N(s))$.

The task is to find a "local maximum", which is an $s \in S$ such that $s = N(s)$.

2.3 The Karchmer-Wigderson game

Karchmer and Wigderson found a characterization of the circuit depth of boolean functions using communication complexity.

▶ **Theorem 1.** [7] *The minimal depth of a circuit computing a boolean function $f(x_1, \ldots, x_n)$ is equal to the minimal number of bits that two players need to communicate in the worst case in the following game. Player I gets an input u such that $f(u) = 0$ and Player II gets an input v such that $f(v) = 1$. By sending messages, they should determine an index i such that $u_i \neq v_i$.*

This theorem also holds for *partial* boolean functions.

3 Razborov's characterization of circuit complexity

Note that the task in the Karchmer-Wigderson game can be viewed as a communication complexity analogue of search problems. Given u and v such that $f(u) = 0$ and $f(v) = 1$, there always exists an index i such that $u_i \neq v_i$ and the task is to find such an i. In communication complexity theory one speaks about computing *relations*, but the analogy with search problems is more appropriate.

When analogues of the usual complexity concepts are defined in communication complexity, $O(\log n)$ communication bits correspond to polynomial time. Thus in the Karchmer-Wigderson Theorem the boolean functions of $\mathbf{NC_1/poly}$ are characterized by the communication complexity class corresponding to search problems solvable in polynomial time. Razborov came up with the idea to characterize $\mathbf{P/poly}$, a probably larger class of functions, by a communication complexity analogue of a probably larger class of polynomial search problems. He showed that such a class of search problems is **PLS**.

To state his theorem we have to translate the definition of **PLS** into a communication complexity problem. Let a partial boolean function $f(x_1, \ldots, x_n)$ be given. Again, Player I gets an input u such that $f(u) = 0$ and Player II gets an input v such that $f(v) = 1$. So the predicate F and the functions N and c will now depend on u and v. But we are not interested in the computational complexity of these functions, only in their communication complexity. Roughly speaking, we want, for every $s \in S$, the communication complexity of computing $F(s, u, v), N(s, u, v)$ and $c(s, u, v)$ to be small.

The goal of the players is again to determine an index i such that $u_i \neq v_i$, but now they want to do it by first computing a local maximum. Therefore, we need another function $p : S \to \{1, 2, \ldots, n\}$ that tells the players the index, given a local maximum s. The function only depends on $s \in S$, thus it does not play any role in defining the complexity of the problem. What however does play an important role is the size of the set of feasible solutions, $\{s \in S;\ F(s, u, v)\}$.

We say that the (f, F, N, c, p) is a **PLS** *communication protocol* if for every u, v such that $f(u) = 0$ and $f(v) = 1$ and every local maximum s (with respect to the parameters u, v), the number $p(s)$ is an index such that $u_{p(s)} \neq v_{p(s)}$.

The complexity of a protocol (f, F, N, c, p) is defined to be the number

$$C = \left| \bigcup_{f(u)=0, f(v)=1} \{s \in S;\ F(s, u, v)\} \right| \cdot 2^{2CC(F,c)+CC(N)},$$

where $2CC(F, c)$ is the maximal communication complexity of computing simultaneously $F(s, u, v)$ and $c(s, u, v)$ and $CC(N)$ is the maximal communication complexity of computing $N(s, u, v)$.

This is a rather technical definition that enables Razborov to state his theorem in a strong form. However, if we were only interested in the communication complexity analogue of **PLS**, we would only require that $|S|$ be of polynomial size, which would correspond to the exponential size of the search space in the usual **PLS**, and that $CC(F), CC(c), CC(N)$ be $O(\log n)$, which would correspond to F, c, N being computable in polynomial time. Then, clearly, the number C would be bounded by a polynomial in n.

▶ **Theorem 2.** [17] *For a given partial boolean function f, the smallest complexity C of* **PLS** *communication protocols (f, F, N, c, p) is, up to a constant factor, equal to the circuit complexity of f.*

We will consider a special case of the theorem that has a more transparent proof. We restrict the above protocols (f, F, N, c, p) as follows.

1. To compute $F(s, u, v)$, the players only need to send one bit to each other independently on each other.
2. For every $s \in S$, either $N(s, u, v) = s$ independently of u, v, or there are two elements $s_0, s_1 \in S$ and one assigned player such that, given u, v, $N(s) \in \{s_0, s_1\}$, and the assigned player knows $N(s)$; thus the player only needs to send one bit to the other player.
3. c only depends on s, not on u, v.

We will call such protocols *restricted* **PLS** *communication protocols.* Essentially, these are protocols in which the players need to send the minimal possible number of bits. Note that condition 1. can be stated more explicitly as follows.

1. There are two predicates $F_I(s, u)$ and $F_{II}(s, v)$ such that $F(s, u, v) \equiv F_I(s, u) \wedge F_{II}(s, v)$.

▶ **Lemma 3.** *The smallest $|S|$ in restricted protocols for f is equal to the circuit complexity of f.*

We will sketch the proof of this lemma.

1. First, assume a circuit D computing f is given. Define S to be the nodes of the circuit, except that we have to rename the output node of the circuit to $\bar{0}$. Given u, v such that $f(u) = 0$, $f(v) = 1$, the predicate $F(s, u, v)$ is defined to be true if, for the function f_s computed at the gate s, $f_s(u) \neq f_s(v)$. The cost function c is an arbitrary antimonotone function from the DAG of the circuit to natural numbers. Given a node s, if it is an input, then $N(s) = s$, otherwise s_0 and s_1 are its input nodes. The assigned player is Player I if the gate at s is $\wedge$ and Player II if the gate is $\vee$. For an input node s labeled by x_i or $\neg x_i$, $p(s) = i$; otherwise it is defined arbitrarily.

We leave the verification of the properties to the reader.

2. Consider a protocol (f, F, N, c, p). Let $U = \{u;\ f(u) = 0\}$ and $V = \{v;\ f(v) = 1\}$. Let $s \in S$ be feasible for some $u \in U$ and $v \in V$, which means that it satisfies $F(s, u, v)$. The condition 1. concerning F says that the set $\{(u, v) \in U \times V;\ F(s, u, v)\}$ is a combinatorial rectangle $U_s \times V_s$, where $U_s = \{u \in U;\ F_I(s, u)\}$ and $V_s = \{v \in V;\ F_{II}(s, v)\}$. We will construct a circuit whose nodes will be the elements of S such that, for every $s \in S$, the function f_s computed at the node s will satisfy

$$f_s(u) = 0 \text{ and } f_s(v) = 1, \text{ for all } u \in U_s \text{ and } v \in V_s. \tag{1}$$

If s is such that $N(s) = s$ and $p(s) = i$, then we label s by x_i or $\neg x_i$. If s is feasible for some u, v, then exactly one of the two labels is correct; otherwise we do not care. To see that only one label is correct, suppose that s is feasible for u and v, and $u_i = 0$ and $v_i = 1$. Then the label should be x_i. It is not possible that s is feasible for some u' and v' such

that $u'_i = 1$ and $v'_i = 0$. If it were, we would also have $F(s, u', v)$, because it is equivalent to $F_I(s, u') \wedge F_{II}(s, v)$. But this is impossible, because $u'_i = v_i$.

Now suppose that s is such that $N(s, u, v) \in \{s_0, s_1\}$ is associated with Player I. It may still happen that, for some $i = 0, 1$, $N(s, u, v) = s_i$ for all $(u, v) \in U_s \times V_s$. In such a case we just join s by a wire to s_i and $f_s = f_{s_i}$. Otherwise we label s by $\wedge$ and connect it to s_0 and s_1. The verification that condition (1) is preserved reduces to proving the following two implications:

$$F(s, u, v) \to F_I(s_0, u) \vee F_I(s_1, u),$$
$$F(s, u, v) \to F_{II}(s_0, v) \wedge F_{II}(s_1, v).$$

The first implication follows trivially from $F(s, u, v) \to F(N(s), u, v) \to F(s_0, u, v) \vee F(s_1, u, v)$. To prove the second one, suppose w.l.o.g. that $N(s, u, v) = s_0$. Since $N(s, u, v)$ only depends on u and since the value s_1 is also possible, there must be some $u' \in U_s$ such that $N(s, u', v) = s_1$. Thus we have $F(s_0, u, v) \wedge F(s_1, u', v)$, whence $F_{II}(s_0, v) \wedge F_{II}(s_1, v)$. ◀

Theorem 2 can be now proved from Lemma 3 by showing that general communication protocols can be reduced to the restricted protocols of Lemma 3. For example, if the players need $k > 1$ bits to compute N, we introduce, for every s, $2^k - 1$ new vertices and replace the arrows $s \to s_0, s \to s_1$ by a tree. Instead of going directly from s to s_0 or s_1, the players will go to these vertices in k steps. Similarly, we have to replace each vertex by 2^ℓ vertices if the feasibility predicate F needs $\ell > 1$ communication bits to be decided, etc.

4 Feasible interpolation

Suppose $\phi(\bar{x}, \bar{y}) \vee \psi(\bar{x}, \bar{z})$ is a tautology where $\bar{x}$ is the string of propositional variables that ϕ and ψ share and $\bar{y}$ and $\bar{z}$ are disjoint strings. Suppose we substitute a string of truth values $\bar{a}$ for $\bar{x}$. Then the terms in the tautology $\phi(\bar{a}, \bar{y}) \vee \psi(\bar{a}, \bar{z})$ have disjoint sets of propositional variables, hence either $\phi(\bar{a}, \bar{y})$ is a tautology, or $\psi(\bar{a}, \bar{z})$ is a tautology, or both. Thus such tautologies give us a computational problem: given $\bar{a}$, determine which of the two formulas $\phi(\bar{a}, \bar{y})$ or $\psi(\bar{a}, \bar{z})$ is a tautology. In general, this problem is not in $\mathbf{P}$, unless $\mathbf{P} = \mathbf{NP} \cap \mathbf{coNP}$. But what if we not only know that $\phi(\bar{x}, \bar{y}) \vee \psi(\bar{x}, \bar{z})$ is a tautology, but also have a proof of it? Krajíček's important discovery is that in some cases we can solve this task if we have a proof. Exactly when this is possible depends on the proof system.

▶ **Definition 4.** We say that a propositional proof system P has the feasible interpolation property, if there exists a polynomial time algorithm A such that given a P-proof d of $\phi(\bar{x}, \bar{y}) \vee \psi(\bar{x}, \bar{z})$ and an assignment $\bar{a}$ for the common variables $\bar{x}$,

1. if $A(d, \bar{a}) = 0$, then $\phi(\bar{a}, \bar{y})$ is a tautology,
2. if $A(d, \bar{a}) = 1$, then $\psi(\bar{a}, \bar{z})$ is a tautology.

▶ **Theorem 5.** [9] *The Resolution proof system has the feasible interpolation property.*

We will reproduce Krajíček's proof and show how it can be done only using Lemma 3. The basic idea is to use the given refutation d to define a communication search problems in the sense of Razborov and from it to construct a circuit C that satisfies 1. and 2. of the theorem. Since C is constructed in polynomial time from d, the construction gives us the polynomial algorithm A.

Let d be a Resolution derivation of the empty clause from two sets of clauses $\Phi(\bar{x}, \bar{y})$ and $\Psi(\bar{x}, \bar{z})$, where $\bar{x}$ are the common variables of the two sets.

Let f be the partial function defined by:

$$f(\bar{u}) = 0 \quad \text{if} \quad \exists \bar{y} \bigwedge \Phi(\bar{u}, \bar{y}),$$
$$f(\bar{v}) = 1 \quad \text{if} \quad \exists \bar{z} \bigwedge \Psi(\bar{v}, \bar{z}),$$

and otherwise undefined. For every $\bar{u}$ such that $f(\bar{u}) = 0$ we choose a $\bar{w}_u$ such that $\bigwedge \Phi(\bar{u}, \bar{y})$ is true, and similarly we define $\bar{t}_v$ for every $\bar{v}$ such that $f(\bar{v}) = 1$.

The search space S is the set of clauses of the proof d. The initial element $\bar{0} \in S$ is the empty clause. The cost function c is the distance from the empty clause.

Given a clause Σ, the feasibility predicate $F(\Sigma, \bar{u}, \bar{v})$ is satisfied, if both assignments $\bar{u}, \bar{w}_u \bar{t}_v$ and $\bar{v}, \bar{w}_u, \bar{t}_v$ falsify Σ.

If Σ is an initial clause, we put $N(\Sigma) = \Sigma$; the definition of $p(\Sigma)$ is irrelevant, because such a Σ is never feasible. Suppose, now, that $\Sigma = \Gamma, \Delta$ and it was derived from clauses $\Sigma_0 = \Gamma, p$ and $\Sigma_1 = \Delta, \neg p$. We distinguish several cases according to to which set of variables p belongs.

1. If $p = y_j$, then we assign Σ to Player I and $N(\Sigma, \bar{u}, \bar{v}) = \Sigma_{(w_u)_j}$.
2. If $p = z_l$, then we assign Σ to Player II and $N(\Sigma, \bar{u}, \bar{v}) = \Sigma_{(t_v)_l}$.
3. If $p = x_i$, then the players tell each other the values u_i and v_i. Then
   a. if $u_i = v_i$, then $N(\Sigma, \bar{u}, \bar{v}) = \Sigma_{u_i}$;
   b. otherwise $N(\Sigma, \bar{u}, \bar{v}) = \Sigma$ and $p(\Sigma) = i$.

This is a protocol in the sense of Razborov, so one can apply his Theorem 2. However, one can also easily reduce it to the simpler Lemma 3. Note that the above protocol almost satisfies the restrictions needed in Lemma 3. Only in 3. the players have to send two bits instead of one. This can be rectified as follows.

Add to the search space S also all x_i and $\neg x_i$, if they are not already present. Put $N(x_i) = x_i$, $N(\neg x_i) = \neg x_i$ and $p(x_i) = p(\neg x_i) = i$. Define $F(x_i, \bar{u}, \bar{v})$ to be true if $u_i = 0$ and $v_i = 1$ and dually for $\neg x_i$.

For $\Sigma, \Sigma_0, \Sigma_1$ as above such that the resolved variable is $p = x_i$, add two new vertices Σ'_0 and Σ'_1 to S. Associate Σ with Player I and both Σ'_0 and Σ'_1 with Player II. Then define:

- $N(\Sigma, \bar{u}, \bar{v}) = \Sigma'_{u_j}$;
- if $u_i = v_i$, put $N(\Sigma'_\nu, \bar{u}, \bar{v}) = \Sigma_\nu$, for $\nu = 0, 1$;
- if $u_i \neq v_i$, put $N(\Sigma'_0, \bar{u}, \bar{v}) = \neg x_i$ and $N(\Sigma'_1, \bar{u}, \bar{v}) = x_i$;
- $F_I(x_i, \bar{u}) \equiv u_i = 0$ and $F_{II}(x_i, \bar{v}) \equiv v_i = 1$;
- $F_I(\neg x_i, \bar{u}) \equiv u_i = 1$ and $F_{II}(\neg x_i, \bar{v}) \equiv v_i = 0$;
- $F_I(\Sigma'_i, \bar{u}) \equiv F_I(\Sigma, \bar{u}) \wedge N(\Sigma, \bar{u}, \bar{v}) = i$ for $i = 0, 1$;
- $F_{II}(\Sigma'_i, v) \equiv F_{II}(\Sigma, v)$ for $i = 0, 1$.

We leave the verification of the properties to the reader. ◀

We have shown that there exists a circuit C with the properties required by the theorem whose size is at most $2n + 3|d|$, where $|d|$ denotes the number of clauses in the Resolution proof d. In [15] we gave a different, more direct proof of Theorem 5. In our construction the number of vertices in the circuit C is at most $2n + |d|$; however, the circuit uses on top of the usual gates $\wedge$ and $\vee$ (together with literals x_i and $\neg x_i$) also the ternary gate *selector*. If the selector gates are replaced by circuits in the basis $\wedge, \vee$, we get the same bound as above $2n + 3|d|$. But not only that: the circuits are identical.

5 From communication protocols to proofs

We are studying three things: circuits, **PLS** communication protocols and Resolution proofs. We have shown how to construct a protocol from a circuit, a circuit from a protocol and a protocol from a proof. Further, this gives us a construction of a circuit from proof, by transitivity, but a direct construction was shown in [15]. To complete the picture, it remains to construct proofs from protocols and circuits. We will only show the construction of a proof from a protocol. The other remaining construction follows by transitivity, but, certainly, a direct construction is also possible. To simplify the presentation, we will only consider protocols for the Karchmer-Wigderson games and restricted **PLS** protocols.

▶ **Proposition 6.** Let P be a protocol in the form of a Karchmer-Wigderson game for computing a partial boolean function $f(\bar{x})$ that uses k communication bits. Then one can construct a tautology of the form $\phi(\bar{x},\bar{y}) \vee \psi(\bar{x},\bar{z})$, with $\phi(\bar{x},\bar{y})$ and $\psi(\bar{x},\bar{z})$ in DNF, and a Resolution proof d of it such that

1. the size of the proof is $|d| = O(|2^k|)$ and
2. the formulas $\phi(\bar{x},\bar{y})$ and $\psi(\bar{x},\bar{z})$ define a partial boolean function that is at least as much defined as f, which means
   a. if $f(\bar{u}) = 0$, then there exists $\bar{w}$ such that $\phi(\bar{u},\bar{w})$ is false, and
   b. if $f(\bar{v}) = 1$, then there exists $\bar{t}$ such that $\psi(\bar{v},\bar{t})$ is false.

Furthermore, the proof has the form of a tree.

To prove the proposition, consider all possible situations that may appear when playing according to the protocol P. We will inductively assign a clause to each of them. For the initial situation when they start, we take the empty clause. Suppose we are in a situation s with a clause Σ and Player I is to speak. Then we choose a new variable y_i and assign $\Sigma \vee y_i$ to the situation after Player I sent the bit 0, respectively, $\Sigma \vee \neg y_i$ to the situation after Player I sent the bit 1. If it is Player II to speak we do the same with a variable z_j instead of y_i. Suppose that the game ends in a situation where the players learn that the ith bit of Player I is 0 while the ith bit of Player II is 1. Let Σ be the clause assigned to this situation, let Σ_I, respectively Σ_{II}, be the subclause of Σ consisting of y_is, respectively z_js. Then we introduce two clauses

$$\Sigma_I \vee \neg x_i \quad \text{and} \quad \Sigma_{II} \vee x_i.$$

If it is 1 and 0, we take x_i in the first clause and $\neg x_i$ in the second.

One can see immediately that the clauses form a Resolution refutation. Let us check condition 2.(a). Let $\bar{u}$ such that $f(u) = 0$ be given. We want to find an assignment that makes all initial clauses made of x_is and y_js false. Given u, the protocol P determines how Player I plays in each situation, so we can set the values w according to what the protocol says. Since the protocol is correct, the clause $\Sigma_I \vee \neg x_i$ (respectively $\Sigma_I \vee x_i$) must be satisfied.

Again, we leave the details to the reader. ◀

▶ **Proposition 7.** Let P be a restricted **PLS** communication protocol for computing a partial boolean function $f(\bar{x})$ with a search space S. Then one can construct a tautology of the form $\phi(\bar{x},\bar{y}) \vee \psi(\bar{x},\bar{z})$, with $\phi(\bar{x},\bar{y})$ and $\psi(\bar{x},\bar{z})$ in DNF, and its Resolution proof d such that

1. the size of the proof is $|d| = O(|S|)$ and
2. the formulas $\phi(\bar{x},\bar{y})$ and $\psi(\bar{x},\bar{z})$ define a partial boolean function that is at least as much defined as f (in the sense of the previous proposition).

The idea of the proof is essentially the same: we introduce a variable for every possible bit sent by the players and describe the dependences between them by clauses. Here are more details.

For every node s in the search space S, we introduce variables $y_{F,s}$ and $z_{F,s}$. The meaning is that s is feasible iff $y_{F,s} \wedge z_{F,s}$ is true. If s is assigned to Player I (respectively Player II), we also introduce $y_{N,s}$ (respectively $z_{N,s}$) for the neighborhood function N.

Let s be such that $N(s) = s$ and suppose that the function p determines that $u_i = 0$ and $v_i = 1$ (where $\bar{u}$ is the input of Player I and $\bar{v}$ is the input of Player II).[1] Then we introduce the clauses

$$y_{F,s} \to \neg x_i \quad \text{and} \quad z_{F,s} \to x_i. \tag{2}$$

Again, if it is 1 and 0, then we switch the negation at x_i.

Let s be such that $N(s) \neq s$, $N(s) \in \{s_0, s_1\}$ and s is assigned to Player I. Then we introduce the following clauses:

$$\begin{aligned}
&(y_{F,s} \wedge y_{N,s}) \to y_{F,s_0},\\
&(y_{F,s} \wedge \neg y_{N,s}) \to y_{F,s_1},\\
&z_{F,s} \to z_{F,s_0},\\
&z_{F,s} \to z_{F,s_1}.
\end{aligned}$$

If s is assigned to Player II we take the same clauses with ys and zs switched.

Finally we take two clauses for the initial node $\bar{0}$:

$$y_{F,\bar{0}} \quad \text{and} \quad z_{F,\bar{0}}.$$

In order to derive the empty clause from these clauses, first derive $\neg y_{F,s} \vee \neg z_{F,s}$ from every pair of clauses (2). Then continue deriving such clauses for all $s \in S$ (going in the direction of decreasing cost). Once we have $\neg y_{F,\bar{0}} \vee \neg z_{F,\bar{0}}$, we resolve with the last two clauses to obtain the empty clause.

The verification of the condition 2. of the proposition is the same as in the previous proof. ◀

Combining the proof of Theorem 2 and Proposition 7 we get a construction of a tautology and its Resolution proof from a function and its circuit. Unfortunately, this is not the converse to feasible interpolation. In particular, it does not give us a reduction of proving lower bounds on circuit complexity to proving lower bounds on the length of proofs. Such a reduction would require a construction that, for a given partial boolean function f and a suitable representation of f by a tautology τ, would transform any boolean circuit for f into a proof of τ. It seems rather unlikely that such a reduction is possible. We only have the following trivial observation.

▶ **Corollary 8.** *Let $f_n(x_1, \ldots, x_n)$ $n = 1, 2, \ldots$, be a sequence of booelan functions. Suppose that there is no sequence of DNF tautologies $\phi_n(\bar{x}, \bar{y}) \vee \psi_n(\bar{x}, \bar{z})$ such that they represent the boolean functions f_n in the sense of Proposition 6 and have polynomial size Resolution proofs. Then f_n do not have polynomial size circuits.*

[1] According to the definition p only tells the index, but we have already noticed that the actual values are also determined.

6 Generalizations

A natural question connected with Razborov's characterization of circuit complexity is: *What happens when we replace* **PLS** *by a different class of polynomial search problems?* One can consider subclasses of **PLS** and get characterizations of classes of circuits with certain restrictions. This line of research may be interesting, but we are interested in the opposite direction: replacing **PLS** by larger classes of polynomial search problems.

Our motivation is to generalize the feasible interpolation property and show that it holds true for stronger proof systems. There are results [13, 4, 3] showing that if certain bit commitment schemas are secure, then sufficiently strong proof systems do not have the feasible interpolation property. Therefore we need a concept of computation that is stronger than boolean circuits, but not too strong, otherwise it would not provide us with any new information about the complexity of the interpolation problem. Generalizations of Razborov's theorem seems to be the right place to look for such concepts.

The classes of search problems that we want to use instead of **PLS** are those that characterize provably total polynomial search problems in fragments of Bounded Arithmetic. The fragments form a hierarchy, denoted by $T_2^0, T_2^1, T_2^2, \ldots$, where, roughly speaking, T_2^n is a theory that is based on the induction axioms for sets of complexity Σ_n^p from the Polynomial Time Hierarchy. These theories have a tight connection to bounded depth Frege propositional proof systems. The provably total polynomial search problems of T_2^0 are solvable in polynomial time; for T_2^1 they belong to **PLS**. Characterizations for higher fragments were found relatively recently, see e.g. [14, 2, 16] (but there are more papers about it).

Very recently, working with Neil Thapen, we obtained a kind of generalization of boolean circuits that can be used for a generalized feasible interpolation theorem for depth 2 Frege systems. This result is too fresh to be included in this paper. We will present it on the conference.

The negative results about feasible interpolation apply to bounded depth Frege of a sufficiently large depth, but it is not clear where the border is. In particular, we do not have an argument implying that depth 2 Frege systems do not have the feasible interpolation property. This is one more reason for studying generalizations.

How do the search problems come into play?

Suppose that

$$\bigwedge \Phi(\bar{x}, \bar{y}) \ \wedge \ \bigwedge \Psi(\bar{x}, \bar{z}) \tag{3}$$

is not satisfiable. This is equivalent to saying that, for every $\bar{u}, \bar{w}, \bar{v}, \bar{t}$, if $\bar{u}, \bar{w}$ satisfies $\bigwedge \Phi(\bar{x}, \bar{y})$ and $\bar{v}, \bar{t}$ satisfies $\bigwedge \Psi(\bar{x}, \bar{z})$, then $\bar{u} \neq \bar{v}$. The last condition is equivalent to the statement that $u_i \neq v_i$ for some i. So the following is a search problem associated with a formula of the form (3):

> *Given a proof d in a proof system P of unsatisfiability of (3) and assignments $\bar{u}, \bar{w}, \bar{v}, \bar{t}$ such that $\bigwedge \Phi(\bar{u}, \bar{w}) \ \wedge \ \bigwedge \Psi(\bar{v}, \bar{t})$ is true, find i such that $u_i \neq v_i$.*

For this problem to be nontrivial, we have to scale it up to an exponentially large structure. Think of the formula, the proof and the assignments as being exponentially large. For example, we can represent the assignments $\bar{u}$ and $\bar{v}$ as *boolean functions* $\bar{u}, \bar{v} : \{0,1\}^n \to \{0,1\}$. In Bounded Arithmetic this is formalized by second order theories, [5].

In order to prove that the search problem above always has a solution we need to prove that the proof system is sound. The strength of the theory needed to prove it depends on the strength of the proof system P; the stronger the proof system is the stronger the theory must be.

For proving soundness of Resolution, T^1_2 suffices. All provably total polynomial search problems in T^1_2 are reducible to **PLS**. Hence the problem above is reducible to a **PLS** problem. The functions and the predicate in such a **PLS** problem can be viewed as polynomial time algorithms that use $\bar{u}, \bar{v}$ as oracles. What is only important for us that they only ask a polynomial number of queries. If we now scale it down from the exponential domain to the polynomial one, we see that to compute these functions and this predicate we only need a logarithmic number of communication bits. This gives us the **PLS** communication protocol.

In a similar fashion we can associate classes of search problems with depth d Frege systems for $d > 1$.

Acknowledgment I would like to thank to Jan Krajíček and Neil Thapen for their comments on the draft of this paper.

References

1 M. Ajtai. The complexity of the Pigeonhole Principle, Combinatorica Volume 14, Number 4, (1994), 417-433.

2 A. Beckmann and S. Buss. Characterizing Definable Search Problems in Bounded Arithmetic via Proof Notations. Preliminary manuscript, 2009

3 M. L. Bonet, C. Domingo, R. Gavaldá, A. Maciel, and T. Pitassi. Non-Automatizability of Bounded-Depth Frege Proofs, Computational Complexity 13:1-2 (2004), 47-68.

4 M. L. Bonet, T. Pitassi, R. Raz. On Interpolation and Automatization for Frege Systems, SIAM Journal of Computing 29(6) (2000), pp. 1939-1967

5 S. Cook and P. Nguyen. Logical Foundations of Proof Complexity, ASL and Cambridge University Press, 2010.

6 A. Haken. The intractability of resolution. Theoretical Computer Science, 39:297–308, 1985.

7 M. Karchmer, A. Wigderson. Monotone Circuits for Connectivity require Super-Logarithmic Depth, SIAM Journal on Discrete Mathematics, vol. 3, no. 2, pp. 255-65, 1990.

8 Lower Bounds to the Size of Constant-Depth Propositional Proofs, J. of Symbolic Logic, 59(1), (1994), pp.73-86.

9 J. Krajíček. Interpolation theorems, lower bounds for proof systems, and independence results for bounded arithmetic, Journal of Symbolic Logic, 62(2), (1997), pp.457-486.

10 J. Krajíček. Tautologies from pseudo-random generators, Bulletin of Symbolic Logic, 7(2), (2001), pp.197-212.

11 J. Krajíček. A form of feasible interpolation for constant depth Frege systems , Journal of Symbolic Logic, 75(2), (2010), pp. 774-784.

12 J. Krajíček. Forcing with random variables and proof complexity, London Mathematical Society Lecture Note Series, No.382, Cambridge University Press, to appear in 2010, 280pp.

13 J. Krajíček and P. Pudlák. Some Consequences of Cryptographical Conjectures for S^1_2 and EF, Information and Computation, Volume 140, Number 1, January 1998 , pp. 82-94(13)

14 J. Krajíček, A. Skelley and N. Thapen. NP search problems in low fragments of bounded arithmetic, in Journal of Symbolic Logic, Vol 72:2, 2007

15 P. Pudlák: Lower bounds for resolution and cutting planes proofs and monotone computations, Journal of Symb. Logic 62(3), 1997, pp.981-998.

16 P. Pudlák and N. Thapen. Alternating minima and maxima, Nash equilibria and Bounded Arithmetic, preprint, 2009.

17 A. A. Razborov. Unprovability of Lower Bounds on the Circuit Size in Certain Fragments of Bounded Arithmetic, in Izvestiya of the Russian Academy of Science, mathematics, Vol. 59, No 1, 1995, pages 201-224.

Recent Progress and Open Problems in Algorithmic Convex Geometry*

Santosh S. Vempala

School of Computer Science
Algorithms and Randomness Center
Georgia Tech, Atlanta, USA 30332
vempala@gatech.edu

Abstract

This article is a survey of developments in algorithmic convex geometry over the past decade. These include algorithms for sampling, optimization, integration, rounding and learning, as well as mathematical tools such as isoperimetric and concentration inequalities. Several open problems and conjectures are discussed on the way.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.42

Contents

* This work was supported by NSF awards AF-0915903 and AF-0910584.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 42–64

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1 Introduction

Algorithmic convex geometry is the study of the algorithmic aspects of convexity. While convex geometry, closely related to finite-dimensional functional analysis, is a rich, classical field of mathematics, it has enjoyed a resurgence since the late 20th century, coinciding with the development of theoretical computer science. These two fields started coming together in the 1970's with the development of geometric algorithms for convex optimization, notably the Ellipsoid method. The ellipsoid algorithm uses basic notions in convex geometry with profound consequences for computational complexity, including the polynomial-time solution of linear programs. This development heralded the use of more geometric ideas, including Karmarkar's algorithm and interior point methods. It brought to the forefront fundamental geometric questions such as *rounding*, i.e., applying affine transformations to sets in Euclidean space to make them easier to handle. The ellipsoid algorithm also resulted in the striking application of continuous geometric methods for the solution of discrete optimization problems such as submodular function minimization.

A further startling development occurred in 1989 with the discovery of a polynomial method for estimating the volume of a convex body. This was especially surprising in the light of an exponential lower bound for any deterministic algorithm for volume estimation. The crucial ingredient in the volume algorithm was an efficient procedure to sample nearly uniform random points from a convex body. Over the past two decades the algorithmic techniques and analysis tools for sampling and volume estimation have been greatly extended and refined. One noteworthy aspect here is the development of isoperimetric inequalities that are of independent interest as purely geometric properties and lead to new directions in the study of convexity. Efficient sampling has also lead to alternative polynomial algorithms for convex optimization.

Besides optimization, integration and sampling, our focus problems in this survey are *rounding* and *learning*. Efficient algorithms for the first three problems rely crucially on rounding. Approximate rounding can be done using the ellipsoid method and with a tighter guarantee using random sampling. Learning is a problem that generalizes integration. For example, while we know how to efficiently compute the volume of a polytope, learning one efficiently from random samples is an open problem. For general convex bodies, volume computation is efficient, while learning requires a superpolynomial number of samples (volume computation uses a membership oracle while learning gets only random samples).

This survey is not intended to be comprehensive. There are numerous interesting developments in convex geometry, related algorithms and their growing list of applications and connections to other areas (e.g., data privacy, quantum computing, statistical estimators etc.) that we do not cover here, being guided instead by our five focus problems.

1.1 Basic definitions

Recall that a subset S of $\mathbb{R}^n$ is convex if for any two points $x, y \in S$, the interval $[x, y] \subseteq S$. A function $f : \mathbb{R}^n \to \mathbb{R}_+$ is said to be *logconcave* if for any two points $x, y \in \mathbb{R}^n$ and any $\lambda \in [0, 1]$,

$$f(\lambda x + (1 - \lambda)y) \geq f(x)^{\lambda} f(y)^{1-\lambda}.$$

The indicator function of a convex set and the density function of a Gaussian distribution are two canonical examples of logconcave functions.

A density function $f : \mathbb{R}^n \to \mathbb{R}_+$ is said to be *isotropic*, if its centroid is the origin, and its covariance matrix is the identity matrix. In terms of the associated random variable X,

this means that

$$\mathsf{E}(X) = 0 \quad \text{and} \quad \mathsf{E}(XX^T) = I.$$

This condition is equivalent to saying that for every unit vector $v \in \mathbb{R}^n$,

$$\int_{\mathbb{R}^n} (v^T x)^2 f(x)\, dx = 1.$$

We say that f is C-isotropic, if

$$\frac{1}{C} \leq \int_{\mathbb{R}^n} (v^T x)^2 f(x)\, dx \leq C$$

for every unit vector v. A convex body is said to be isotropic if the uniform density over it is isotropic. For any full-dimensional distribution D with bounded second moments, there is an affine transformation of space that puts it in isotropic position, namely, if

$$z = \mathsf{E}_D(X) \text{ and } A = \mathsf{E}((X - z)(X - z)^T)$$

then $y = A^{-\frac{1}{2}}(X - z)$ has an isotropic density.

For a density function $f : \mathbb{R}^n \to \mathbb{R}_+$, we let π_f be the measure associated with it. The following notions of distance between two distributions P and Q will be used: The *Total variation distance* of P and Q is

$$d_{tv}(P, Q) = \sup_{A \in \mathcal{A}} |P(A) - Q(A)|.$$

The χ-*squared distance* of P with respect to Q is

$$\chi^2(P, Q) = \int_{\mathbb{R}}^n \left(\frac{dP(u) - dQu}{dQ(u)} \right)^2 dQ(u) = \int_{\mathbb{R}}^n \frac{dP(u)}{dQ(u)}\, dP(u) - 1$$

The first term in the last expression is also called the L_2 distance of P w.r.t. Q. Finally, P is said to be M-*warm* w.r.t. Q if

$$M = \sup_{A \in \mathcal{A}} \frac{P(A)}{Q(A)}.$$

A discrete-time *Markov chain* is defined using a triple $(K, \mathcal{A}, \{P_u : u \in K\})$ along with a starting distribution Q_0, where K is the state space, $\mathcal{A}$ is a set of measurable subsets of K and P_u is a measure over K, as a sequence of elements of K, $w_0, w_1, \ldots$, where w_0 is chosen from Q_0 and each subsequent w_i is chosen from $P_{w_{i-1}}$. Thus, the choice of w_{i+1} depends only on w_i and is independent of $w_0, \ldots, w_{i-1}$. It is easy to verify that a distribution Q is stationary iff for every $A \in \mathcal{A}$,

$$\int_A P_u(K \setminus A)\, dQ(u) = \int_{K \setminus A} P_u(A)\, dQ(u).$$

The *conductance* of a subset A is defined as

$$\phi(A) = \frac{\int_A P_u(K \setminus A)\, dQ(u)}{\min\{Q(A), Q(K \setminus A)\}}$$

and the conductance of the Markov chain is

$$\phi = \min_A \phi(A) = \min_{0 < Q(A) \leq \frac{1}{2}} \frac{\int_A P_u(K \setminus A)\, dQ(u)}{Q(A)}.$$

The following weaker notion of conductance will also be useful. For any $0 \leq s < \frac{1}{2}$, the *s-conductance* of a Markov chain is defined as

$$\phi_s = \min_{A: s < Q(A) \leq \frac{1}{2}} \frac{\int_A P_u(K \setminus A)\, dQ(u)}{Q(A) - s}.$$

We let $\mathsf{Var}_f(g)$ denote the variance of a real-valued function g w.r.t. a density function f. The unit ball in $\mathbb{R}^n$ by B_n. We use $O^*(.)$ to suppress error parameters and terms that are polylogarithmic in the leading terms.

2 Problems

We now state our focus problems more precisely and mention the current bounds on their complexity. Algorithms achieving these bounds are discussed in a Section 4. As input to these problems, we typically assume a general function oracle, i.e., access to a real-valued function as a blackbox. The complexity will be measured by both the number of oracle calls and the number of arithmetic operations.

2.1 Optimization

Input: an oracle for a function $f : \mathbb{R}^n \to \mathbb{R}_+$; a point x_0 with $f(x_0) > 0$; an error parameter ε.
Output: a point x such that

$$f(x) \geq \max f - \varepsilon.$$

Linear optimization over a convex set is the special case when $f = e^{-c^T x}\chi^K(x)$ where $c^T x$ is a linear function to be minimized and K is the convex set. This is equivalent to a membership oracle for K in this case. Given only access to the function f, for any logconcave f, the complexity of optimization is $\mathrm{poly}(n, \log(1/\varepsilon))$ by either an extension of the Ellipsoid method to handle membership oracles [21], or Vaidya's algorithm [60] or a reduction to sampling [6, 25, 45]. This line of work began with the breakthrough result of Khachiyan [34] showing that the Ellipsoid method proposed by Yudin and Nemirovskii [65] is polynomial for explicit linear programs. The current best time complexity is $O^*(n^{4.5})$ achieved by a reduction to logconcave sampling [45]. For optimization of an explicit function over a convex set, the oracle is simply membership in the convex set. A stronger oracle is typically available, namely a *separation* oracle that gives a hyperplane that separates the query point from the convex set, in the case when the point lies outside. Using a separation oracle, the ellipsoid algorithm has complexity $O^*(n^2)$ while Vaidya's algorithm and that of Bertsimas and Vempala have complexity $O^*(n)$.

The current frontier for polynomial-time algorithms is when f is a logconcave function in $\mathbb{R}^n$. Slight generalizations, e.g., linear optimization over a star-shaped body is NP-hard, even to solve approximately [49, 9].

2.2 Integration/Counting

Input: an oracle for a function $f : \mathbb{R}^n \to \mathbb{R}_+$; a point x_0 with $f(x_0) > 0$; an error parameter ε.
Output: a real number A such that

$$(1-\varepsilon)\int f \leq A \leq (1+\varepsilon)\int f.$$

Dyer, Frieze and Kannan [15, 16] gave a polynomial-time randomized algorithm for estimating the volume of a convex body (the special case above when f is the indicator function of a convex body) with complexity $\text{poly}(n, 1/\varepsilon, \log(1/\delta))$, where δ is the probability that the algorithm's output is incorrect. This was extended to integration of logconcave functions by Applegate and Kannan [3] with an additional dependence of the complexity on the Lipschitz parameter of f. Following a long line of improvements, the current best complexity is $O^*(n^4)$ using a variant of simulated annealing, quite similar to the algorithm for optimization [45].

In the discrete setting, the natural general problem is when f is a distribution over the integer points in a convex set, e.g., f is 1 only for integer points in a convex body K. While there are many special cases that are well-studied, e.g., perfect matchings of a graph [23], not much is known in general except a roundness condition [30].

2.3 Learning

Input: random points drawn from an unknown distribution in $\mathbb{R}^n$ and their labels given by an unknown function $f : \mathbb{R}^n \to \{0, 1\}$; an error parameter ε.
Output: A function $g : \mathbb{R}^n \to \{0, 1\}$ such that $\mathsf{P}(g(x) \neq f(x)) \leq \varepsilon$ over the unknown distribution.

The special case when f is an unknown linear function can be solved by using any linear classifier that correctly labels a sample of size

$$\frac{C}{\varepsilon}\left(n \log \frac{1}{\varepsilon} + \log \frac{1}{\delta}\right),$$

where C is a constant. Such a classifier can be learned by using any efficient algorithm for linear programming. The case when f is a polynomial threshold function of bounded degree can also be learned efficiently essentially by reducing to the case of linear thresholds via a linearization of the polynomial.

2.4 Sampling

Input: an oracle for a function $f : \mathbb{R}^n \to \mathbb{R}_+$; a point x_0 with $f(x_0) > 0$; an error parameter ε.
Output: a random point x whose distribution D is within ε total variation distance of the distribution with density proportional to f.

As the key ingredient of their volume algorithm [16], Dyer, Frieze and Kannan gave a polynomial-time algorithm for the case when f is the indicator function over a convex body. The complexity of the sampler itself was $\text{poly}(n, 1/\varepsilon)$. Later, Applegate and Kannan [3] generalized this to smooth logconcave functions with complexity polynomial in $n, 1/\varepsilon$ and the Lipschitz parameter of the logconcave function. In [45] the complexity was improved to $O^*(n^4)$ and the dependence on the Lipschitz parameter was removed.

2.5 Rounding.

We first state this problem for a special case.
Input: a membership oracle for a convex body K; a point $x_0 \in K$; an error parameter ε.
Output: a linear transformation A and a point z such that $K' = A(K - z)$ satisfies one of the following:

- Exact sandwiching:

$$B_n \subseteq K' \subseteq RB_n. \tag{1}$$

- Second moment sandwiching: For every unit vector u,

$$1 - \varepsilon \leq \mathsf{E}_{K'}((u \cdot x)^2) \leq 1 + \varepsilon.$$

- Approximate sandwiching:

$$\mathrm{vol}(B_n \cap K') \geq \frac{1}{2}\mathrm{vol}(B_n) \text{ and } \mathrm{vol}(K' \cap RB_n) \geq \frac{1}{2}\mathrm{vol}(K'). \tag{2}$$

When the input is a general density function, the second moment condition extends readily without any change to the statement.

The classical Löwner-John theorem says that for any convex body K, the ellipsoid E of maximal volume contained in K has the property that $K \subseteq nE$ (and $K \subseteq \sqrt{n}E$ if K is centrally symmetric). Thus by using the transformation that makes this ellipsoid a ball, one achieves $R = n$ in the first condition above (exact sandwiching). Moreover, this is the best possible as shown by the simplex. However, computing the ellipsoid of maximal volume is hard. Lovász [41] showed how to compute an ellipsoid that satisfies the containment with $R = n^{3/2}$ using the Ellipsoid algorithm. This remains the best-known deterministic algorithm. A randomized algorithm can achieve $R = n(1 + \varepsilon)$ for any $\varepsilon > 0$ using a simple reduction to random sampling [27]. In fact, all that one needs is $n \cdot C(\varepsilon)$ random samples [8, 57, 54, 1], and then the transformation to be computed is the one that puts the sample in isotropic position.

3 Geometric inequalities and conjectures

We begin with some fundamental inequalities. For two subsets A, B of $\mathbb{R}^n$, their Minkowski sum is

$$A + B = \{x + y : x \in A, y \in B\}.$$

The Brunn-Minkowski inequality says that if A, B and $A + B$ are measurable, compact subsets of $\mathbb{R}^n$, then

$$\mathrm{vol}(A + B)^{\frac{1}{n}} \geq \mathrm{vol}(A)^{\frac{1}{n}} + \mathrm{vol}(B)^{\frac{1}{n}}. \tag{3}$$

The following extension is the Prékopa-Leindler inequality: for any three functions $f, g, h : \mathbb{R}^n \to \mathbb{R}_+$, satisfying

$$h(\lambda x + (1 - \lambda)y) \geq f(x)^\lambda g(x)^{1-\lambda}$$

for every $\lambda \in [0, 1]$,

$$\int_{\mathbb{R}^n} h(x)\,dx \geq \left(\int_{\mathbb{R}^n} f(x)\,dx\right)^\lambda \left(\int_{\mathbb{R}^n} g(x)\,dx\right)^{1-\lambda}. \tag{4}$$

The product and the minimum of two logconcave functions are also logconcave. We also have the following fundamental properties [12, 39, 55, 56].

▶ **Theorem 1.** *All marginals and the distribution function of a logconcave function are logconcave. The convolution of two logconcave functions is logconcave.*

In the rest of this section, we describe inequalities about rounding, concentration of mass and isoperimetry. These inequalities play an important role in the analysis of algorithms for our focus problems. Many of the inequalities apply to logconcave functions. The latter represent the limit to which methods that work for convex bodies can be extended. This is true for all the algorithms we will see with a few exceptions, e.g., an extension of isoperimetry and sampling to star-shaped bodies. We note here that Milman [50] has shown a general approximate equivalence between concentration and isoperimetry over convex domains.

3.1 Rounding

We next describe a set of properties related to affine transformations. The first one below is from [27].

▶ **Theorem 2.** *[27] Let K be a convex body in $\mathbb{R}^n$ in isotropic position. Then,*

$$\sqrt{\frac{n+1}{n}}B_n \subseteq K \subseteq \sqrt{n(n+1)}B_n.$$

Thus, in terms of the exact sandwiching condition (1), isotropic position achieves a factor of n, and this ratio of the radii of the outer and inner ball, n, is the best possible as shown by the n-dimensional simplex. A bound of $O(n)$ was earlier established by Milman and Pajor [51].

Sandwiching was extended to logconcave functions in [48] as follows.

▶ **Theorem 3.** *[48] Let f be a logconcave density in isotropic position. Then for any $t > 0$, the level set $L(t) = \{x : f(x) \geq t\}$ contains a ball of radius $\pi_f(L(t))/e$. Also, the point $z = \arg\max f$ satisfies $\|z\| \leq n + 1$.*

For general densities, it is convenient to define a rounding parameter as follows. For a density function f, let $R^2 = \mathsf{E}_f(\|X - \mathsf{E}X\|^2)$ and r be the radius of the largest ball contained in the level set of f of measure $1/8$. Then R/r is the rounding parameter and we say a function is *well-rounded* if $R/r = O(\sqrt{n})$. By the above theorem, any logconcave density in isotropic position is well-rounded.

An isotropic transformation can be estimated for any logconcave function using random samples drawn from the density proportional to f. In fact, following the work of Bourgain [8] and Rudelson [57], Adamczak et al [1] showed that $O(n)$ random points suffice to achieve, say, 2-isotropic position with high probability.

▶ **Theorem 4.** *[1] Let $x_1, \ldots x_m$ be random points drawn from an isotropic logconcave density f. Then for any $\varepsilon \in (0, 1), t \geq 1$, there exists $C(\varepsilon, t)$ such that if $m \geq C(\varepsilon, t)n$,*

$$\left\| \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T - I \right\|_2 \leq \varepsilon$$

with probability at least $1 - e^{-ct\sqrt{n}}$. Moreover,

$$C(\varepsilon, t) = C\frac{t^4}{\varepsilon^2}\log^2\frac{2t^2}{\varepsilon^2}$$

for absolute constants c, C suffices.

The following conjecture was alluded to in [44].

▶ Conjecture 5. There exists a constant C such that for any convex body K in $\mathbb{R}^n$, there exists an ellipsoid E that satisfies approximate sandwiching:

$$\mathrm{vol}(E \cap K) \geq \frac{1}{2}\mathrm{vol}(E); \mathrm{vol}(K \cap C\log nE) \geq \frac{1}{2}\mathrm{vol}(K).$$

3.2 Measure and concentration

The next inequality was proved by Grünbaum [22] (for the special case of the uniform density over a convex body).

▶ **Lemma 6.** *[22] Let $f : \mathbb{R}^n \to \mathbb{R}_+$ be a logconcave density function, and let H be any halfspace containing its centroid. Then*

$$\int_H f(x)\,dx \geq \frac{1}{e}.$$

This was extended to hyperplanes that come close to the centroid in [6, 48].

▶ **Lemma 7.** *[6, 48] Let $f : \mathbb{R}^n \to \mathbb{R}_+$ be an isotropic logconcave density function, and let H be any halfspace within distance t from its centroid. Then*

$$\int_H f(x)\,dx \geq \frac{1}{e} - t.$$

The above lemma easily implies a useful concentration result.

▶ **Theorem 8.** *Let f be a logconcave density in $\mathbb{R}^n$ and z be the average of m random points from π_f. If H is a halfspace containing z,*

$$\mathsf{E}\left(\pi_f(H)\right) \geq \left(\frac{1}{e} - \sqrt{\frac{n}{m}}\right).$$

A strong radial concentration result was shown by Paouris [54].

▶ **Lemma 9.** *[54] Let K be an isotropic convex body. Then, for any $t \geq 1$,*

$$\mathsf{P}(\|X\| \geq ct\sqrt{n}) \leq e^{-t\sqrt{n}}$$

where c is an absolute constant.

This implies that all but an exponentially small fraction of the mass of an isotropic convex body lies in a ball of radius $O(\sqrt{n})$.

The next inequality is a special case of a more general inequality due to Brascamp and Lieb. It also falls in a family called Poincaré-type inequalities. Recall that $\mathsf{Var}_\mu(f)$ denote the variance of a real-valued function f w.r.t. a density function μ.

▶ **Theorem 10.** *Let γ be the standard Gaussian density in $\mathbb{R}^n$. Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function. Then*

$$\mathsf{Var}_\gamma(f) \leq \int_{\mathbb{R}^n} \|\nabla f\|^2\,d\gamma.$$

A smooth function here is one that is locally Lipschitz.

An extension of this inequality to logconcave restrictions was developed in [63], eliminating the need for smoothness and deriving a quantitative bound. In particular, the lemma shows how variances go down when the standard Gaussian is restricted to a convex set.

▶ **Theorem 11.** *[63] Let g be the standard Gaussian density function in $\mathbb{R}^n$ and $f : \mathbb{R}^n \to \mathbb{R}_+$ be any logconcave function. Define the function h to be the density proportional to their product, i.e.,*

$$h(x) = \frac{f(x)g(x)}{\int_{\mathbb{R}^n} f(x)g(x)\,dx}.$$

Then, for any unit vector $u \in \mathbb{R}^n$,

$$\mathsf{Var}_h(u \cdot x) \leq 1 - \frac{e^{-b^2}}{2\pi}$$

where the support of f along u is $[a_0, a_1]$ and $b = \min\{|a_0|, |a_1|\}$.

The next conjecture is called the *variance hypothesis* or *thin shell* conjecture.

▶ Conjecture 12 (Thin shell). For X drawn from a logconcave isotropic density f in $\mathbb{R}^n$,

$$\mathsf{Var}_f(\|X\|^2) \leq Cn$$

We note here that for an isotropic logconcave density,

$$\sigma_n^2 = \mathsf{E}_f((\|X\| - \sqrt{n})^2) \leq \frac{1}{n}\mathsf{Var}_f(\|X\|^2) \leq C\sigma_n^2,$$

i.e., the deviation of $\|X\|$ from $\sqrt{n}$ and the deviation of $\|X\|^2$ from $\mathsf{E}(\|X\|^2) = n$ are closely related. Thus, the conjecture implies that most of the mass of a logconcave distribution lies in shell of *constant* thickness.

A bound of $\sigma_n \leq \sqrt{n}$ is easy to establish for any isotropic logconcave density. Klartag showed that $\sigma_n = o(\sqrt{n})$ [36, 37], a result that can be interpreted as a central limit theorem (since the standard deviation is now a smaller order term than the expectation). The current best bound on σ_n is due to Fleury [19] who showed that $\sigma_n \leq n^{3/8}$. We state his result below.

▶ **Lemma 13.** *[19] For any isotropic logconcave measure μ,*

$$\mathsf{P}\left(|\|x\| - \sqrt{n}| \geq tn^{\frac{3}{8}}\right) \leq Ce^{-ct}.$$

where c, C are constants.

3.3 Isoperimetry

The following theorem is from [14], improving on a theorem in [43] by a factor of 2. For two subsets S_1, S_2 of $\mathbb{R}^n$, we let $d(S_1, S_2)$ denote the minimum Euclidean distance between them.

▶ **Theorem 14.** *[14] Let S_1, S_2, S_3 be a partition into measurable sets of a convex body K of diameter D. Then,*

$$\mathrm{vol}(S_3) \geq \frac{2d(S_1, S_2)}{D} \min\{\mathrm{vol}(S_1), \mathrm{vol}(S_2)\}.$$

A limiting (and equivalent) version of this inequality is the following: Let $\partial S \cap K$ denote the interior boundary of S w.r.t. K. For any subset S of a convex body of diameter D,

$$\mathrm{vol}_{n-1}(\partial S \cap K) \geq \frac{2}{D} \min\{\mathrm{vol}(S), \mathrm{vol}(K \setminus S)\},$$

i.e., the surface area of S inside K is large compared to the volumes of S and $K \setminus S$. This is in direct analogy with the classical isoperimetric inequality, which says that the surface area to volume ratio of any measurable set is at least the ratio for a ball. Henceforth, for a density function f, we use $\Phi_f(S)$ to denote the isoperimetric ratio of a subset S, i.e.,

$$\Phi_f(S) = \frac{\pi_f(\partial S)}{\min\{\pi_f(S), 1 - \pi_f(S)\}}$$

and

$$\Phi_f = \inf_{S \subseteq \mathbb{R}^n} \phi(S).$$

Theorem 14 can be generalized to arbitrary logconcave measures.

▶ **Theorem 15.** *Let $f : \mathbb{R}^n \to \mathbb{R}_+$ be a logconcave function whose support has diameter D and let π_f be the induced measure. Then for any partition of $\mathbb{R}^n$ into measurable sets S_1, S_2, S_3,*

$$\pi_f(S_3) \geq \frac{2d(S_1, S_2)}{D} \min\{\pi_f(S_1), \pi_f(S_2)\}.$$

The conclusion can be equivalently stated as $\Phi_f \geq 2/D$. For Euclidean distance this latter statement, applied to the subset S_1 essentially recovers the above lower bound on $\pi_f(S_3)$.

Next we ask if logconcave functions are the limit of such isoperimetry. Theorem 15 can be extended to s-concave functions in $\mathbb{R}^n$ for $s \geq -1/(n-1)$ with only a loss of a factor of 2 [10]. A nonnegative function f is s-concave if $f(x)^s$ is a concave function of s. Logconcave functions correspond to $s \to 0$. The only change in the conclusion is by a factor of 2 on the RHS.

▶ **Theorem 16.** *[10] Let $f : \mathbb{R}^n \to \mathbb{R}_+$ be an s-concave function with $s \geq -1/(n-1)$ and with support of diameter D and let π_f be the induced measure. Then for any partition of $\mathbb{R}^n$ into measurable sets S_1, S_2, S_3,*

$$\pi_f(S_3) \geq \frac{d(S_1, S_2)}{D} \min\{\pi_f(S_1), \pi_f(S_2)\}.$$

This theorem is tight, in that there exist s-concave functions for $s \leq -1/(n-1-\varepsilon)$ for any $\varepsilon > 0$ whose isoperimetric ratio is exponentially small in εn.

Logconcave and s-concave functions are natural extensions of convex bodies. It is natural to ask what classes of nonconvex sets/functions might have good isoperimetry. In recent work [9], the isoperimetry of star-shaped bodies was studied. A star-shaped set S is one that contains at least one point x such that for any $y \in S$, the segment $[x, y]$ is also in S. The set of all such points x for which lines through x have convex intersections with S is called its kernel K_S. The kernel is always a convex set.

▶ **Theorem 17.** *[9] Let S_1, S_2, S_3 be a partition into measurable sets of a star-shaped body S of diameter D. Let $\eta = \mathrm{vol}(K_S)/\mathrm{vol}(S)$. Then,*

$$\mathrm{vol}(S_3) \geq \frac{d(S_1, S_2)}{4\eta D} \min\{\mathrm{vol}(S_1), \mathrm{vol}(S_2)\}.$$

In terms of diameter, Theorem 14 is the best possible, as shown by a cylinder. A more refined inequality is obtained in [27, 48] using the average distance of a point to the center of gravity (in place of diameter). It is possible for a convex body to have much larger diameter than average distance to its centroid (e.g., a cone). In such cases, the next theorem provides a better bound.

▶ **Theorem 18.** *[27] Let f be a logconcave density in $\mathbb{R}^n$ and π_f be the corresponding measure. Let z_f be the centroid of f and define $M(f) = \mathsf{E}_f(|x - z_f|)$. Then, for any partition of $\mathbb{R}^n$ into measurable sets S_1, S_2, S_3,*

$$\pi_f(S_3) \geq \frac{\ln 2}{M(f)} d(S_1, S_2) \pi_f(S_1) \pi_f(S_2).$$

For an isotropic density, $M(f)^2 \leq \mathsf{E}_f(|x - z_f|^2) = n$ and so $M(f) \leq \sqrt{n}$. The diameter could be unbounded (e.g., an isotropic Gaussian). Thus, if A_f is the covariance matrix of the logconcave density f, the inequality can be re-stated as:

$$\pi_f(S_3) \geq \frac{c}{\sqrt{\mathsf{Tr}(A_f)}} d(S_1, S_2)\pi_f(S_1)\pi_f(S_2).$$

A further refinement is called the KLS hyperplane conjecture [27]. Let $\lambda_1(A)$ be the largest eigenvalue of a symmetric matrix A.

▶ Conjecture 19 (KLS). Let f be a logconcave density in $\mathbb{R}^n$. There is an absolute constant c such that

$$\Phi_f \geq \frac{c}{\sqrt{\lambda_1(A_f)}}.$$

The KLS conjecture implies the thin shell conjecture (Conj. 12). Even the thin shell conjecture would improve the known isoperimetry of isotropic logconcave densities due to the following result of Bobkov [7].

▶ **Theorem 20.** *[7] For any logconcave density function f in $\mathbb{R}^n$,*

$$\Phi_f \geq \frac{c}{\mathsf{Var}(\|X\|^2)^{1/4}}.$$

Combining this with Fleury's bound 13, we get that for an isotropic logconcave function in $\mathbb{R}^n$,

$$\Phi_f \geq cn^{-7/16}. \tag{5}$$

This is slightly better than the bound of $cn^{-1/2}$ implied by Theorem 18 since $\mathsf{E}(\|X\|^2) = n$ for an isotropic density.

The next conjecture is perhaps the most well-known in convex geometry, and is called the slicing problem, the isotropic constant or simply the hyperplane conjecture.

▶ Conjecture 21 (Slicing). There exists a constant C, such that for an isotropic logconcave density f in $\mathbb{R}^n$,

$$f(0) \leq C^n.$$

In fact, a result of Ball [4] shows that it suffices to prove such a bound for the case of f being the uniform density over an isotropic convex body. In this case, the conjecture says the volume of an isotropic convex body in $\mathbb{R}^n$ is at least c^n for some constant c. The current best bound on $L_n = \sup_f f(0)^{1/n}$ is $O(n^{1/4})$ [8, 35].

Recently, Eldan and Klartag [18] have shown that the slicing conjecture is implied by the thin shell conjecture (and therefore by the KLS conjecture as well, a result that was earlier shown by Ball).

▶ **Theorem 22.** *[18] There exists a constant C such that*

$$L_n = \sup_f f(0)^{1/n} \leq C\sigma_n = C \sup_f \sqrt{\mathsf{E}_f((\|X\| - \sqrt{n})^2)}$$

where the suprema range over isotropic logconcave density functions.

We conclude this section with a discussion of another direction in which classical isoperimetry has been extended. The *cross-ratio distance* (also called Hilbert metric) between two points u, v in a convex body K is computed as follows: Let p, q be the endpoints of the chord in K through u and v such that the points occur in the order p, u, v, q. Then

$$d_K(u, v) = \frac{|u - v||p - q|}{|p - u||v - q|} = (p : v : u : q).$$

where $(p : v : u : q)$ denotes the classical cross-ratio. We can now define the cross-ratio distance between two sets S_1, S_2 as

$$d_K(S_1, S_2) = \min\{d_K(u, v) \, : \, u \in S_1, v \in S_2\}.$$

The next theorem was proved in [42] for convex bodies and extended to logconcave densities in [47].

▶ **Theorem 23.** *[42] Let f be a logconcave density in $\mathbb{R}^n$ whose support is a convex body K and let π_f be the induced measure. Then for any partition of $\mathbb{R}^n$ into measurable sets S_1, S_2, S_3,*

$$\pi_f(S_3) \geq d_K(S_1, S_2)\pi_f(S_1)\pi_f(S_2).$$

All the inequalities so far are based on defining the distance between S_1 and S_2 by the *minimum* over pairs of some notion of pairwise distance. It is reasonable to think that perhaps a much sharper inequality can be obtained by using some *average* distance between S_1 and S_2. Such an inequality was proved in [46], leading to a substantial improvement in the analysis of hit-and-run.

▶ **Theorem 24.** *[46] Let K be a convex body in $\mathbb{R}^n$. Let $f : K \to \mathbb{R}_+$ be a logconcave density with corresponding measure π_f and $h : K \to \mathbb{R}_+$, an arbitrary function. Let S_1, S_2, S_3 be any partition of K into measurable sets. Suppose that for any pair of points $u \in S_1$ and $v \in S_2$ and any point x on the chord of K through u and v,*

$$h(x) \leq \frac{1}{3} \min(1, d_K(u, v)).$$

Then

$$\pi_f(S_3) \geq \mathsf{E}_f(h(x)) \min\{\pi_f(S_1), \pi_f(S_2)\}.$$

The coefficient on the RHS has changed from a "minimum" to an "average". The weight $h(x)$ at a point x is restricted only by the minimum cross-ratio distance between pairs u, v from S_1, S_2 respectively, such that x lies on the line between them (previously it was the overall minimum). In general, it can be much higher than the minimum cross-ratio distance between S_1 and S_2.

3.4 Localization

Most of the known isoperimetry theorems can be proved using the localization lemma of Lovász and Simonovits [44, 27]. It reduces inequalities in high dimension to 1-dimensional inequalities by a sequence of bisections and a limit argument. To prove that an inequality is true, one assumes it is false and derives a contradiction in the 1-dimensional case.

▶ **Lemma 25.** *[44] Let $g, h : \mathbb{R}^n \to \mathbb{R}$ be lower semi-continuous integrable functions such that*

$$\int_{\mathbb{R}^n} g(x)\,dx > 0 \quad \text{and} \quad \int_{\mathbb{R}^n} h(x)\,dx > 0.$$

Then there exist two points $a, b \in \mathbb{R}^n$ and a linear function $\ell : [0,1] \to \mathbb{R}_+$ such that

$$\int_0^1 \ell(t)^{n-1} g((1-t)a + tb)\,dt > 0 \quad \text{and} \quad \int_0^1 \ell(t)^{n-1} h((1-t)a + tb)\,dt > 0.$$

The points a, b represent an interval A and one may think of $l(t)^{n-1}dA$ as the cross-sectional area of an infinitesimal cone with base area dA. The lemma says that over this cone truncated at a and b, the integrals of g and h are positive. Also, without loss of generality, we can assume that a, b are in the union of the supports of g and h.

A variant of localization was used in [9], where instead of proving an inequality for *every* needle, one instead averages over the set of needles produced by the localization procedure. Indeed, each step of localization is a bisection with a hyperplane and for an inequality to hold for the original set, it often suffices for it hold "on average" over the partition produced rather than for both parts separately. This approach can be used to prove isoperimetry for star-shaped sets (Theorem 17) or to recover Bobkov's isoperimetric inequality (Lemma 20).

We state here a strong version of a conjecture related to localization, which if true implies the KLS hyperplane conjecture.

▶ Conjecture 26. Let f be a logconcave density function and $\mathcal{P}$ be any partition of $\mathbb{R}^n$ such that each part is convex. Then there exists a constant C such that

$$\sum_{\mathcal{P}} \sigma_f(P)^2 \pi_f(P) \leq C\sigma_f^2$$

where σ_f^2 is the largest variance of f along any direction and $\sigma_f(P)^2$ is the largest variance of f restricted to the set P.

We have already seen that the conjecture holds when f is a Gaussian with $C = 1$ (Theorem 11).

4 Algorithms

In this section, we describe the current best algorithms for the five focus problems and several open questions. These algorithms are related to each other, e.g., the current best algorithms for integration and optimization (in the membership oracle model) are based on sampling, the current best algorithms for sampling and rounding proceed in tandem etc.

In fact, all five problems have an intimate relationship with random sampling. Integration (volume estimation), optimization and rounding are solved by reductions to random sampling; in the case of integration, this is the only known efficient approach. For rounding, the sampling approach gives a better bound than the current best alternatives. As for the learning problem, its input is a random sample.

The Ellipsoid method has played a central role in the development of algorithmic convex geometry. Following Khachiyan's polynomial bound for linear programs [34], the generalization to convex optimization [21, 53, 32] has been a powerful theoretical tool. Besides optimization, the method also gave an efficient rounding algorithm [41] achieving a sandwiching ratio of $n^{1.5}$, and a polynomial-time algorithm for PAC-learning linear threshold functions. It is a major open problem to PAC-learn an intersection of *two* halfspaces.

Several polynomial-time algorithms have been proposed for linear programming and convex optimization, including: Karmarkar's algorithm [31] for linear programming, interior-point methods for convex optimization over sets with *self-concordant barriers* [52], polynomial implementation of the perceptron algorithm for linear [13] and conic program [5], simplex-with-rescaling for linear programs [33] and the random walk method [6] that requires only membership oracle access to the convex set. The field of convex optimization continues to be very active, both because of its many applications but also due to the search for more practical algorithms that work on larger inputs.

One common aspect of all the known polynomial-time algorithms is that they use use some type of repeated rescaling of space to effectively make the convex set "more round", leading to a complexity that depends on the accuracy to which the output is computed. In principle, such a dependence can be avoided for linear programming and it is a major open problem to find a strongly polynomial algorithm for solving linear programs.

We note here that although there have been many developments in continuous optimization algorithms in the decades since the ellipsoid method, and in applications to combinatorial problems, there has been little progress in the past two decades on general integer programming, with the current best complexity being $n^{O(n)}$ from Kannan's improvement [26] of Lenstra's algorithm [40].

4.1 Geometric random walks

Sampling is achieved by rapidly mixing geometric random walks. For an in-depth survey of this topic, up-to-date till 2005, the reader is referred to [62]. Here we outline the main results, including developments since then.

To sample from a density proportional to a function $f : \mathbb{R}^n \to \mathbb{R}_+$, we set up a Markov chain whose state space is $\mathbb{R}^n$ and stationary distribution has density proportional to f. At a point x, one step of the chain picks a *neighbor* y of x from some distribution P_x that depends only on x, where P_x is defined in some efficiently sampleable manner, and then we move to y with some probability or stay at x. For example, for uniformly sampling a convex body K, the *ball walk* defines the neighbors of x as all points within a fixed distance δ from x, and a random neighbor y is accepted as long as y is also in K. For sampling a general density, one could use the same neighbor set, and modify the probability of transition to $\min\{1, f(y)/f(x)\}$. This rejection mechanism is called the Metropolis filter and the Markov chain itself is the ball walk with a Metropolis filter.

Another Markov chain that has been successfully analyzed is called *hit-and-run*. At a point x, we pick a uniform random line l passing x, then a random point on the line l according to the density induced by the target distribution along l. In the case of uniformly sampling a convex body, this latter distribution is simply the uniform distribution on a chord; for a logconcave function it is the one-dimensional density proportional to the target density f along the chosen line l. This process has the advantage of not needing a step-size parameter δ.

For ease of analysis, we typically assume that the Markov chain is *lazy*, i.e., it stays put with probability 1/2 and attempts a move with probability 1/2. Then, it follows that the distribution of the current point converges to a unique stationary distribution assuming the conductance of the Markov chain is nonzero. The rate of convergence is approximately determined by the conductance as given by the following theorem due to Lovász and Simonovits [44], extending earlier work by Diaconis and Stroock [11], Alon [2] and Jerrum and Sinclair [58].

▶ **Theorem 27.** *[44] Let Q be the stationary distribution of a lazy Markov chain with Q_0 being its starting distribution and Q_t the distribution of the current point after t steps.*

a. Let $M = \sup_A Q_0(A)/Q(A)$. Then,

$$d_{tv}(Q_t, Q) \leq \sqrt{M}\left(1 - \frac{\phi^2}{2}\right)^t.$$

b. Let $0 < s \leq \frac{1}{2}$ and $H_s = \sup\{|Q_0(A) - Q(A)| : Q(A) \leq s\}$. Then,

$$d_{tv}(Q_t, Q) \leq H_s + \frac{H_s}{s}\left(1 - \frac{\phi_s^2}{2}\right)^t.$$

c. For any $\varepsilon > 0$,

$$d_{tv}(Q_t, Q) \leq \varepsilon + \sqrt{\frac{\chi^2(Q_0, Q) + 1}{\varepsilon}}\left(1 - \frac{\phi^2}{2}\right)^t.$$

Thus the main quantity to analyze in order to bound the mixing time is the conductance.

Ball walk. The following bound holds on the conductance of the ball walk [28].

▶ **Theorem 28.** *For any $0 \leq s \leq 1$, we can choose the step-size δ for the ball walk in a convex body K of diameter D so that*

$$\phi_s \geq \frac{s}{200nD}.$$

The proof of this inequality is the heart of understanding the convergence. Examining the definitions of the conductance ϕ and the isoperimetric ratio Φ_Q of the stationary distribution, one sees that they are quite similar. In fact, the main difference is the following: in defining conductance, the weight between two points x and y depends on the probability density of stepping from x to y in one step; for the isoperimetric ratio, the distance is a geometric notion such as Euclidean distance. Connecting these two notions — geometric distance and probabilistic distance — along with isoperimetry bounds leads to the conductance bound above. This is the generic line of proof with each of these components chosen based on the Markov chain being analyzed. For a detailed description, we refer to [62].

Using Theorem 27(b), we conclude that from an M-warm start, the variation distance of Q_t and Q is smaller than ε after

$$t \geq C\frac{M^2}{\varepsilon^2}n^2D^2\ln\left(\frac{2M}{\varepsilon}\right) \tag{6}$$

steps, for some absolute constant C.

The bound of $O(n^2D^2)$ on the mixing rate is the best possible in terms of the diameter, as shown by a cylinder. Here D^2 can be replaced by $\mathsf{E}(\|X - \mathsf{E}X\|^2)$ using the isoperimetric inequality given by Theorem 18. For an isotropic convex body this gives a mixing rate of $O(n^3)$. The current best isoperimetry for convex bodies (5) gives a bound of $O(n^{2.875})$. The KLS conjecture (19) implies a mixing rate of $O(n^2)$, which matches the best possible for the ball walk, as shown for example by an isotropic cube.

Hit-and-run. For hit-and-run, one obtains a qualitatively better bound in the following sense. In the case of the ball walk, the starting distribution heaving affects the mixing rate. It is possible to start at points close to the boundary with very small local conductance, necessitating many attempts to make a single step. Hit-and-run does not have this problem and manages to exponentially accelerate out of any corner. This is captured in the next theorem, using the isoperimetric inequality given by Theorem 24.

▶ **Theorem 29.** *[46] The conductance of hit-and-run in a convex body of diameter D is $\Omega(1/nD)$.*

▶ **Theorem 30.** *[46] Let K be a convex body that contains a unit ball and has centroid z_K. Suppose that $\mathsf{E}_K(|x - z_K|^2) \leq R^2$ and $\chi^2(Q_0, Q) \leq M$. Then after*

$$t \geq Cn^2R^2 \ln^3 \frac{M}{\varepsilon},$$

steps, where C is an absolute constant, we have $d(Q_t, Q)_{tv} \leq \varepsilon$.

The theorem improves on the bound for the ball walk (6) by reducing the dependence on M and ε from polynomial (which is unavoidable for the ball walk) to logarithmic, while maintaining the (optimal) dependence on R and n. For a body in near-isotropic position, $R = O(\sqrt{n})$ and so the mixing time is $O^*(n^3)$. One also gets a polynomial bound starting from *any single* interior point. If x is at distance d from the boundary, then the distribution obtained after one step from x has $\chi^2(Q_1, Q) \leq (n/d)^n$ and so applying the above theorem, the mixing time is $O(n^4 \ln^3(n/d\varepsilon))$.

Theorems 29 and 30 have been extended in [45] to arbitrary logconcave functions.

▶ **Theorem 31.** *[45] Let f be a logconcave density function with support of diameter D and assume that the level set of measure $1/8$ contains a unit ball. Then,*

$$\phi_s \geq \frac{c}{nD \ln(nD/s)}$$

where c is a constant.

This implies a mixing rate that nearly matches the bound for convex bodies.

Affine-invariant walks. In both cases above, the reader will notice the dependence on rounding parameters and the improvement achieved by assuming isotropic position. As we will see in the next section, efficient rounding can be achieved by interlacing with sampling. Here we mention two random walks which achieve the rounding "implicitly" by being affine invariant. The first is a multi-point variant of hit-and-run that can be applied to sampling any density function. It maintains m points $x_1, \ldots, x_m$. For each x_j, it picks a random combination of the current points,

$$y = \sum_{i=1}^{m} \alpha_i (x_i - \overline{x})$$

where the α_i are drawn from $N(0, 1)$; the chosen point x_j is replaced by a random point along this line through x_j in the direction of y. This process in affine invariant and hence one can effectively assume that the underlying distribution is isotropic. The walk was analyzed in [6] assuming a warm start. It is open to analyze the rate of convergence from a general starting distribution.

For polytopes whose facets are given explicitly, Kannan and Narayanan [29] analyzed an affine-invariant process called the *Dikin walk*. At each step, the Dikin walk computes an ellipsoid based on the current point (and the full polytope) and moves to a random point in this ellipsoid. The Dikin ellipsoid at a point x in a polytope $Ax \leq 1$ with m inequalities is defined as:

$$D_x = \{z \in \mathbb{R}^n \,:\, (x-z)^T \sum_{i=1}^{m} \frac{a_i a_i^T}{(1-a_i^T x)^2}(x-z) \leq 1\}.$$

At x, a new point y is sampled from a Dikin ellipsoid and the walk moves to y with prob

$$\min\left\{1, \frac{\mathrm{vol}(D_y)}{\mathrm{vol}(D_x)}\right\}.$$

For polytopes with few facets, this process uses fewer arithmetic operations than the current best bounds for the ball walk or hit-and-run.

Open problems. One open problem for both hit-and-run and the ball walk is to find a single starting point that is as good a start as a warm start. E.g., does one of these walks started at the centroid converge at the same rate as starting from a random point?

The random walks we have considered so far for general convex bodies and density functions rely only on membership oracles or function oracles. Can one do better using a separation oracle? When presented with a point x such an oracle either declares the point is in the convex set K or gives a hyperplane that separates x from K. At least for convex bodies, such an oracle is typically realizable in the same complexity as a membership oracle.

One attractive process based on a separation oracle is the following *reflection walk* with parameter δ: At a point x, we pick a random point y in the ball of radius δ. We move along the straight line from x to y either reaching y or encountering a hyperplane H that separates y from K; in the latter case, we reflect y about H and continue moving towards y.

It is possible that for this process, the number of oracle calls (not the number of Markov chain steps) is only $O^*(nD)$ rather than $O^*(n^2D^2)$, and even $O^*(n)$ for isotropic bodies. It is a very interesting open problem to analyze the reflection walk.

Our next open problem is to understand classes of distributions that can be efficiently sampled by random walks. A recent extension of the convex setting is to sampling star-shaped bodies [9], and this strongly suggests that the full picture is far from clear. One necessary property is good isoperimetry. Can one provide a sufficient condition that depends on isoperimetry and some local property of the density to be sampled? More concretely, for what manifolds with nonnegative curvature (for which isoperimetry is known) can an efficient sampling process be defined?

Our final question in this section is about sampling a discrete subset of a convex body, namely the set of lattice points that lie in the body. In [30], it is shown that this problem can be solved if the body contains a sufficiently large ball, by a reduction to the continuous sampling problem. The idea is that, under this condition, the volume of the body is roughly the same as the number of lattice points and thus sampling the body and rounding to a nearby lattice point is effective. Can this condition be improved substantially? E.g., can one sample lattice points of a near-isotropic convex body? Or lattice points of a body that contains at least half of a ball of radius $\mathrm{O}(\sqrt{n})$?

4.2 Annealing

Rounding, optimization and integration can all be achieved by variants of the same algorithmic technique that one might call *annealing*. The method starts at an “easy" distribution F_0 and

goes through a sequence of distributions $F_1, \ldots, F_m$ where the F_i are chosen so that moving from F_{i-1} to F_i is efficient and F_m is a target distribution. This approach can be traced back to [43]. It was analyzed for linear optimization over convex sets in [25], for volume computation and rounding convex sets in [47] and extended to integration, rounding and maximization of logconcave functions in [45].

For example, in the case of convex optimization, the distribution F_m is chosen so that most of its mass on points whose objective value is at least $(1-\varepsilon)$ times the maximum. Sampling directly from this distribution could be difficult since one begins at an arbitrary point.

For integration of logconcave functions, we define a series of functions, with the final function being the one we wish to integrate and the initial one being a function that is easy to integrate. The distribution in each phase has density proportional to the corresponding function. We use samples from the current distribution to estimate the ratios of integrals of consecutive functions. Multiplying all these ratios and the integral of the initial function gives the estimate for the integral of the target function.

For rounding a logconcave density, as shown by Theorem 4, we need $O(n)$ random samples. Generating these directly from the target density could be expensive since sampling (using a random walk) takes time that depends heavily on how well-rounded the density function is. Instead, we consider again a sequence of distributions, where the first distribution in the sequence is chosen to be easy to sample and consecutive distributions have the property that if F_i is isotropic, then F_{i+1} is near-isotropic. We then sample F_{i+1} efficiently, apply an isotropic transformation and proceed to the next phase.

For both optimization and integration, this rounding procedure is incorporated into the main algorithm to keep the sampling efficient. All three cases are captured in the generic description below.

Annealing

1. For $i = 0, \ldots, m$, define
$$a_i = b\left(1 + \frac{1}{\sqrt{n}}\right)^i \text{ and } f_i(x) = f(x)^{a_i}.$$
2. Let $X_0^1, \ldots, X_0^k$ be independent random points with density proportional to f_0.
3. For $i = 0, \ldots, m-1$: starting with $X_i^1, \ldots, X_i^k$, generate random points $X_{i+1} = \{X_{i+1}^1, \ldots, X_{i+1}^k\}$; update a running estimate g based on these samples; update the isotropy transformation using the samples.
4. Output the final estimate of g.

For optimization, the function f_m is set to be a sufficiently high power of f, the function to be maximized while g is simply the maximum objective value so far. For integration and rounding $f_m = f$, the target function to be integrated or rounded. For integration, the function g starts out as the integral of f_0 and is multiplied by the ratio of $\int f_{i+1} / \int f_i$ in each step. For rounding, g is simply the estimate of the isotropic transformation for the current function.

We now state the known guarantees for this algorithm [45]. The complexity improves on the original $O^*(n^{10})$ algorithm of Applegate and Kannan [3].

▶ **Theorem 32.** *[45] Let f be a well-rounded logconcave function. Given $\varepsilon, \delta > 0$, we can compute a number A such that with probability at least $1 - \delta$,*

$$(1-\varepsilon)\int_{\mathbb{R}^n} f(x)\,dx \le A \le (1+\varepsilon)\int_{\mathbb{R}^n} f(x)\,dx$$

and the number of function calls is

$$O\left(n^4 \log^c \frac{n}{\varepsilon\delta}\right) = O^*(n^4)$$

where c is an absolute constant.

Any logconcave function can be put in near-isotropic position in time $O^*(n^4)$ steps, given a point that maximizes f. Near-isotropic position guarantees that f is well-rounded as remarked earlier. When the maximum of f is not known in advance or computable quickly, the complexity is a bit higher and is the same as that of maximization.

▶ **Theorem 33.** *[45] For any well-rounded logconcave function f, given $\varepsilon, \delta > 0$ and a point x_0 with $f(x_0) \ge \beta \max f$, the annealing algorithm finds a point x in $O^*(n^{4.5})$ oracle calls such that with probability at least $1 - \delta$, $f(x) \ge (1-\varepsilon)\max f$ and the dependence on ε, δ and β is bounded by a polynomial in $\ln(1/\varepsilon\delta\beta)$.*

This improves significantly on the complexity of the Ellipsoid method in the memebership oracle model (the latter being $\Omega(n^{10})$). We observe here that in this general oracle model, the upper bound on the complexity of optimization is higher than that of integration. The gap gets considerably higher when we move to star-shaped sets. Integration remains $O^*(n^4/\eta^2)$ where η is the relative measure of the kernel of the star-shaped set, while linear optimization, even approximately is NP-hard. The hardness holds for star-shaped sets with η being any constant [9], i.e., the convex kernel takes up any constant fraction of the set. At one level, this is not so surprising since finding a maximum could require zooming in to a small hidden portion of the set while integration is a more global property. On the other hand, historically, efficient integration algorithms came later than optimization algorithms even for convex bodies and were considerably more challenging to analyze.

The known analysis of sampling-based methods for optimization rely on the current point being nearly random from a suitable distribution, with the distribution modified appropriately at each step. It is conceivable that a random walk type method can be substantially faster if its goal is optimization and its analysis directly measures progress on some distance function, rather than relying on the intermediate step of producing a random sample.

The modern era of volume algorithms began when Lovász asked the question of whether the volume of convex body can be estimated from a random sample (unlike annealing in which the sampling distribution is modified several times). Eldan [17] has shown that this could require a superpolynomial number of points for general convex bodies. On the other hand, it remains open to efficiently compute the volume from a random sample for polytopes with a bounded number of facets.

Annealing has been used to speed up the reduction from counting to sampling for discrete sets as well [59]. Here the traditional reduction with overhead linear in the underlying dimension [24] is improved to one that is roughly the square root of the dimension for estimating a wide class of partition functions. The annealing algorithm as described above has to be extended to be nonadaptive, i.e., the exponents used change in an adaptive manner rather than according to a schedule fixed in advance.

4.3 PCA

The method we discuss in this section is classical, namely Principal Component Analysis (PCA). For a given set of points or a distribution in $\mathbb{R}^n$, PCA identifies a sequence of orthonormal vectors such that for any $k \leq n$, the span of the first k vectors in this sequence is the subspace that minimizes the expected squared distance to the given point set or distribution (among all k-dimensional subsets). The vectors can be found using the Singular Value Decomposition (SVD), which can be viewed as a greedy algorithm that finds one vector at a time. More precisely, given a distribution D in $\mathbb{R}^n$, with $\mathsf{E}_D(X) = 0$, the top principal component or singular vector is a unit vector that maximizes $\mathsf{E}((v^T x)^2)$. For $2 \leq i \leq n$, the i'th principal component is a unit vector that maximizes the same function among all unit vectors that are orthogonal to the first $i-1$ principal components. For a general Gaussian, the principal components are exactly the directions along which the component 1-dimensional Gaussians are generated. Besides the regression property for subspaces, PCA is attractive because it can be computed efficiently by simple, iterative algorithms. Replacing the expected squared distance by a different power leads to an intractable problem. We have already seen one application of PCA, namely to rounding via the isotropic position. Here we take the principal components of the covariance matrix and apply a transformation to make their corresponding singular values all equal to one (and therefore the variance in any direction is the same). The principal components are the unit vectors along the axes of the inertial ellipsoid corresponding to the covariance matrix of a distribution.

We next discuss the application of PCA to the problem of learning an intersection of k halfpsaces in $\mathbb{R}^n$. As remarked earlier, this is an open problem even for $k = 2$. We make two assumptions. First, k is small compared to n and so algorithms that are exponential in k but not n might be tolerable. Second, instead of learning such an intersection from an arbitrary distribution on examples, we assume the distribution is an unknown Gaussian. This setting was considered by Vempala [61, 64] and by Klivans et al [38]. The first algorithm learns an intersection of k halfspaces from any logconcave input distribution using an intersection of $O(k)$ halfspaces and has complexity $(n/\varepsilon)^{O(k)}$. The second learns an intersection of k halfspaces from any Gaussian distribution using a polynomial threshold function of degree $O(\log k/\varepsilon^4)$ and therefore has complexity $n^{O(\log k/\varepsilon^4)}$. Neither of these algorithms is polynomial when k grows with n.

In recent work [63], PCA is used to give an algorithm whose complexity is $\text{poly}(n, k, 1/\varepsilon) + C(k, 1/\varepsilon)$ where $C(.)$ is the complexity of learning an intersection of k halfspaces in $\mathbb{R}^k$. Thus one can bound this using prior results as at most

$$\min\left\{k^{O(\log k/\varepsilon^4)}, (k/\varepsilon)^{O(k)}\right\}.$$

For fixed ε, the algorithm is polynomial for k up to $2^{O(\sqrt{\log n})}$. The algorithm is straightforward: first put the full distribution in isotropic position (using a sample), effectively making the input distribution a standard Gaussian; then compute the *smallest* k principal components of the examples that lie in the intersection of the unknown halfspaces. The main claim in the analysis is that this latter subspace must be close to the span of the normals to the unknown halfspaces. This is essentially due to Lemma 11, which guarantees that the smallest k principal components are in the subspace spanned by the normals, while orthogonal to this subspace, all variances are equal to that of the standard Gaussian. The algorithm and its analysis can be extended to arbitrary convex sets whose normals lie in an unknown k-dimensional subspace. The complexity remains a fixed polynomial in n times an exponential in k.

For general convex bodies, given positive and negative examples from an unknown Gaussian distribution, it is shown in [38] that the complexity is $2^{\tilde{O}(\sqrt{n})}$ (ignoring the dependence on ε, along with a nearly matching lower bound. A similar lower bound of $2^{\Omega(\sqrt{n})}$ holds for learning a convex body given only random points from the body [20]. These constructions are similar to Eldan's [17] and use polytopes with an exponential number of facets. Thus an interesting open problem is to learn a polytope in $\mathbb{R}^n$ with m facets given either uniform random points from it or from a Gaussian. It is conceivable that the complexity is $\text{poly}(m, n, 1/\varepsilon)$. The general theory of VC-dimension already tells us that $O(mn)$ samples suffice. Such an algorithm would of course also give us an algorithm for estimating the volume of a polytope from a set of random points and not require samples from a sequence of distributions.

Acknowledgements. The author is grateful to Daniel Dadush, Elena Grigorescu, Ravi Kannan, Jinwoo Shin and Ying Xiao for helpful comments.

References

1 R. Adamczak, A. Litvak, A. Pajor, and N. Tomczak-Jaegermann. Quantitative estimates of the convergence of the empirical covariance matrix in log-concave ensembles. *J. Amer. Math. Soc.*, 23:535–561, 2010.

2 N. Alon. Eigenvalues and expanders. *Combinatorica*, 6:83–96, 1986.

3 D. Applegate and R. Kannan. Sampling and integration of near log-concave functions. In *STOC '91: Proceedings of the twenty-third annual ACM symposium on Theory of computing*, pages 156–163, New York, NY, USA, 1991. ACM.

4 K. M. Ball. Logarithmically concave functions and sections of convex sets in rn. *Studia Mathematica*, 88:69–84, 1988.

5 A. Belloni, R. M. Freund, and S. Vempala. An efficient rescaled perceptron algorithm for conic systems. *Math. Oper. Res.*, 34(3):621–641, 2009.

6 D. Bertsimas and S. Vempala. Solving convex programs by random walks. *J. ACM*, 51(4):540–556, 2004.

7 S. Bobkov. On isoperimetric constants for log-concave probability distributions. *Geometric aspects of functional analysis, Lect. notes in Math.*, 1910:81–88, 2007.

8 J. Bourgain. Random points in isotropic convex sets. *Convex geometric analysis*, 34:53–58, 1996.

9 K. Chandrasekaran, D. Dadush, and S. Vempala. Thin partitions: Isoperimetric inequalities and a sampling algorithm for star shaped bodies. In *SODA*, pages 1630–1645, 2010.

10 K. Chandrasekaran, A. Deshpande, and S. Vempala. Sampling s-concave functions: The limit of convexity based isoperimetry. In *APPROX-RANDOM*, pages 420–433, 2009.

11 P. Diaconis and D. Stroock. Geometric bounds for eigenvalues of markov chains. *Ann. Appl. Probab.*, 1(1):36–61, 1991.

12 A. Dinghas. Uber eine klasse superadditiver mengenfunktionale vonbrunn-minkowski-lusternik-schem typus. *Math. Zeitschr.*, 68:111–125, 1957.

13 J. Dunagan and S. Vempala. A simple polynomial-time rescaling algorithm for solving linear programs. *Math. Prog.*, 114(1):101–114, 2008.

14 M. E. Dyer and A. M. Frieze. Computing the volume of a convex body: a case where randomness provably helps. In *Proc. of AMS Symposium on Probabilistic Combinatorics and Its Applications*, pages 123–170, 1991.

15 M. E. Dyer, A. M. Frieze, and R. Kannan. A random polynomial time algorithm for approximating the volume of convex bodies. In *STOC*, pages 375–381, 1989.

16 M. E. Dyer, A. M. Frieze, and R. Kannan. A random polynomial-time algorithm for approximating the volume of convex bodies. *J. ACM*, 38(1):1–17, 1991.

17 R. Eldan. A polynomial number of random points does not determine the volume of a convex body. *http://arxiv.org/abs/0903.2634*, 2009.

18 R. Eldan and B. Klartag. Approximately gaussian marginals and the hyperplane conjecture. *http://arxiv.org/abs/1001.0875*, 2010.

19 B. Fleury. Concentration in a thin euclidean shell for log-concave measures. *J. Funct. Anal.*, 259(4):832–841, 2010.

20 N. Goyal and L. Rademacher. Learning convex bodies is hard. In *COLT*, 2009.

21 M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, 1988.

22 B. Grunbaum. Partitions of mass-distributions and convex bodies by hyperplanes. *Pacific J. Math.*, 10:1257–1261, 1960.

23 M. Jerrum, A. Sinclair, and E. Vigoda. A polynomial-time approximation algorithm for the permanent of a matrix with nonnegative entries. *J. ACM*, 51(4):671–697, 2004.

24 M. Jerrum, L. G. Valiant, and V. V. Vazirani. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.*, 43:169–188, 1986.

25 A. T. Kalai and S. Vempala. Simulated annealing for convex optimization. *Math. Oper. Res.*, 31(2):253–266, 2006.

26 R. Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12(3):415–440, 1987.

27 R. Kannan, L. Lovász, and M. Simonovits. Isoperimetric problems for convex bodies and a localization lemama. *Discrete & Computational Geometry*, 13:541–559, 1995.

28 R. Kannan, L. Lovász, and M. Simonovits. Random walks and an $O^*(n^5)$ volume algorithm for convex bodies. *Random Structures and Algorithms*, 11:1–50, 1997.

29 R. Kannan and H. Narayanan. Random walks on polytopes and an affine interior point method for linear programming. In *STOC*, pages 561–570, 2009.

30 R. Kannan and S. Vempala. Sampling lattice points. In *STOC*, pages 696–700, 1997.

31 N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–396, 1984.

32 R. M. Karp and C. H. Papadimitriou. On linear characterization of combinatorial optimization problems. *SIAM J. Comp.*, 11:620–632, 1982.

33 J. A. Kelner and D. A. Spielman. A randomized polynomial-time simplex algorithm for linear programming. In *STOC*, pages 51–60, 2006.

34 L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20:53–72, 1980.

35 B. Klartag. On convex perturbations with a bounded isotropic constant. *Geom. and Funct. Anal.*, 16(6):1274–1290, 2006.

36 B. Klartag. A central limit theorem for convex sets. *Invent. Math.*, 168:91–131, 2007.

37 B. Klartag. Power-law estimates for the central limit theorem for convex sets. *J. Funct. Anal.*, 245:284–310, 2007.

38 A. R. Klivans, Ryan O'Donnell, and R. A. Servedio. Learning geometric concepts via gaussian surface area. In *FOCS*, pages 541–550, 2008.

39 L. Leindler. On a certain converse of Hölder's inequality ii. *Acta Sci. Math. Szeged*, 33:217–223, 1972.

40 H. W. Lenstra. Integer programming with a fixed number of variables. *Math. of Oper. Res.*, 8(4):538–548, 1983.

41 L. Lovász. *An Algorithmic Theory of Numbers, Graphs and Convexity*, volume 50 of *CBMS-NSF Conference Series*. SIAM, 1986.

42 L. Lovász. Hit-and-run mixes fast. *Math. Prog.*, 86:443–461, 1998.

43 L. Lovász and M. Simonovits. On the randomized complexity of volume and diameter. In *Proc. 33rd IEEE Annual Symp. on Found. of Comp. Sci.*, pages 482–491, 1992.

44 L. Lovász and M. Simonovits. Random walks in a convex body and an improved volume algorithm. In *Random Structures and Alg.*, volume 4, pages 359–412, 1993.

45 L. Lovász and S. Vempala. Fast algorithms for logconcave functions: Sampling, rounding, integration and optimization. In *FOCS*, pages 57–68, 2006.

46 L. Lovász and S. Vempala. Hit-and-run from a corner. *SIAM J. Computing*, 35:985–1005, 2006.

47 L. Lovász and S. Vempala. Simulated annealing in convex bodies and an $O^*(n^4)$ volume algorithm. *J. Comput. Syst. Sci.*, 72(2):392–417, 2006.

48 L. Lovász and S. Vempala. The geometry of logconcave functions and sampling algorithms. *Random Structures and Algorithms*, 30(3):307–358, 2007.

49 J. Luedtke, S. Ahmed, and G. L. Nemhauser. An integer programming approach for linear programs with probabilistic constraints. *Math. Program.*, 122(2):247–272, 2010.

50 E. Milman. On the role of convexity in isoperimetry, spectral gap and concentration. *Invent. Math.*, 177(1):1–43, 2009.

51 V. Milman and A. Pajor. Isotropic position and inertia ellipsoids and zonoids of the unit ball of a normed n-dimensional space. *Geometric aspects of Functional Analysis, Lect. notes in Math.*, pages 64–104, 1989.

52 Yu. Nesterov and A. Nemirovski. *Interior-Point Polynomial Algorithms in Convex Programming*. Studies in Applied Mathematics. SIAM, 1994.

53 M. W. Padberg and M. R. Rao. The russian method for linear programming iii: Bounded integer programming. *NYU Research Report*, 1981.

54 G. Paouris. Concentration of mass on convex bodies. *Geometric and Functional Analysis*, 16:1021–1049, 2006.

55 A. Prekopa. Logarithmic concave measures and functions. *Acta Sci. Math. Szeged*, 34:335–343, 1973.

56 A. Prekopa. On logarithmic concave measures with applications to stochastic programming. *Acta Sci. Math. Szeged*, 32:301–316, 1973.

57 M. Rudelson. Random vectors in the isotropic position. *Journal of Functional Analysis*, 164:60–72, 1999.

58 A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82:93–133, 1989.

59 D. Stefankovic, S. Vempala, and E. Vigoda. Adaptive simulated annealing: A near-optimal connection between sampling and counting. In *FOCS*, pages 183–193, Washington, DC, USA, 2007. IEEE Computer Society.

60 P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Math. Prog.*, 73:291–341, 1996.

61 S. Vempala. A random sampling based algorithm for learning the intersection of half-spaces. In *FOCS*, pages 508–513, 1997.

62 S. Vempala. Geometric random walks: A survey. *MSRI Combinatorial and Computational Geometry*, 52:573–612, 2005.

63 S. Vempala. Learning convex concepts from gaussian distributions with PCA. In *FOCS*, 2010.

64 S. Vempala. A random sampling based algorithm for learning the intersection of half-spaces. *JACM, to appear*, 2010.

65 D. B. Yudin and A. S. Nemirovski. Evaluation of the information complexity of mathematical programming problems (in russian). *Ekonomika i Matematicheskie Metody*, 13(2):3–45, 1976.

Playing in stochastic environment: from multi-armed bandits to two-player games

Wiesław Zielonka[1]

1 **LIAFA, Université Paris 7 Denis Diderot, Paris, France**
zielonka@liafa.jussieu.fr

Abstract

Given a zero-sum infinite game we examine the question if players have optimal memoryless deterministic strategies. It turns out that under some general conditions the problem for two-player games can be reduced to the same problem for one-player games which in turn can be reduced to a simpler related problem for multi-armed bandits.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.65

1 Introduction

Activities of a computer system interacting with the environment are often modeled as two-player games with one player representing the system and the other player impersonating the environment. In the worst case analysis we assume that the environment is hostile and then we deal with two-player zero-sum games. Traditionally in verification and in automata theory we use some variants of parity games [7] while the traditional game theory focuses on mean-payoff games, discounted games and total payoff games [1].

Parity games capture qualitative system properties, but sometimes this is not enough and we are interested in finer quantitative analysis. Mean-payoff and total payoff games capture quantitative properties but do not seem really pertinent in the context of computer systems. For these reasons there were recently several attempts to define new quantitative measures or payoffs better suited to the analysis of computer systems. This is an ongoing activity, each such attempt gives rise to a new game (a new payoff mapping).

And the recurrent basic question arising when new games (payoffs) are introduced is if players have optimal strategies. However for a computer scientist the existence of optimal or nearly optimal strategies is not sufficient, we want to be able to implement effectively such strategies and strategies requiring an unbounded memory are unfeasible from the practical point of view. A finite memory can often be incorporated directly into the game and then it is sufficient to answer the simpler question if the players have optimal memoryless strategies. Instead of examining various games one by one with some ad hoc methods it is much more interesting to look for general sufficient conditions guaranteeing the existence of optimal memoryless strategies. Such conditions are useful only if they are robust and can be applied to a sufficiently large class of games.

Our aim is to present such general conditions, we do it first for one-player games (Markov decision processes), next for two-player games.

2 Perfect information stochastic games – basic definitions

2.1 Notation.

For each finite set X, $\mathcal{D}(X)$ is the set of probability measures over X, i.e. it is the set of mappings $p : X \to [0,1]$ such that $\sum_{x\in X} p(x) = 1$. The support of $p \in \mathcal{D}(X)$ is the set

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 65–72

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\{x \in X \mid p(x) > 0\}$.

X^i will denote the set of all finite sequences (words) of length i composed of elements of X, $X^* = \cup_{i=0}^{\infty} X^i$ is the set of all finite words over X, X^ω will stand for the set of all infinite words over X. We endow X^ω with the structure of a topological space with open sets of the form $\bigcup_{u \in L} uX^\omega$ for $L \subseteq X^*$. By $\mathcal{B}(X^\omega)$ we denote the smallest σ-algebra containing all open sets (the Borel σ-algebra). Thus $(X^\omega, \mathcal{B}(X^\omega))$ is a measurable space.

By $\pi_i : X^\omega \to X$, $i \in \mathbb{N}$, we denote the mapping defined as $\pi_i(x_1x_2x_3\ldots) = x_i$, i.e. the mapping giving the ith element of an infinite word.

If we equip X with the σ-algebra $\mathcal{P}(X)$ of all subsets of X then π_i are measurable mappings from $(X^\omega, \mathcal{B}(X^\omega))$ to $(X, \mathcal{P}(X))$ and, given any probability measure P on $(X^\omega, \mathcal{B}(X^\omega))$, $\{\pi_i; i \in \mathbb{N}\}$ becomes a discrete process with values in X. Note that $\mathcal{B}(X^\omega)$ is in fact the smallest σ-algebra such that all π_i are measurable.

2.2 Games and Arenas

Two players, Min and Max, play an infinite game on an arena

$$\mathcal{A} = (\mathbf{S}, \mathbf{A}, \text{source}, \delta, \text{player}) \tag{1}$$

where

- $\mathbf{S}$ is a finite set of states,
- $\mathbf{A}$ is a finite set of actions,
- source : $\mathbf{A} \to \mathbf{S}$ provides for each action $a \in \mathbf{A}$ a state source$(a) \in \mathbf{S}$ called the source of a. Action a can be executed only if the current state is $s = \text{source}(a)$ and then we say that a is available at s. We write $\mathbf{A}(s)$ for the set of actions available at s and we assume that $\mathbf{A}(s) \neq \emptyset$ for each state s.
- The dynamic aspect of $\mathcal{A}$ is described by δ, for each action $a \in \mathbf{A}$ and each state $s \in \mathbf{S}$ $\delta(a, s)$ is the probability of going to a state s if a is executed. It is tacitly assumed that a can be executed only if $\mathcal{A}$ is at the state source(a). For each action $a \in \mathbf{A}$, $\delta(a, \cdot)$ is a probability distribution over $\mathbf{S}$, i.e. $\delta(a, \cdot) \in \mathcal{D}(\mathbf{S})$.
- Finally, player : $\mathbf{S} \to \{\text{Min}, \text{Max}\}$ is a mapping assigning to each state $s \in \mathbf{S}$ the player player(s) controlling s.

The game is played by stages. If at stage $i \in \mathbb{N}$ the game is at state $s_i \in \mathbf{S}$ then player player(s_i) chooses an available action $a_i \in \mathbf{A}(s_i)$ and the game enters a new state s_{i+1} with probability $\delta(a_i, s_{i+1})$.

Let $\mathbf{S}_{\text{Max}} = \{s \in \mathbf{S} \mid \text{player}(s) = \text{Max}\}$ be the set of states controlled by player Max. A strategy σ for player Max is a mapping

$$\sigma : \mathbf{A}^* \times \mathbf{S}_{\text{Max}} \to \mathcal{D}(\mathbf{A})$$

such that $\sigma(h, s) \in \mathcal{D}(\mathbf{A}(s))$ for $h \in \mathbf{A}^*$ and $s \in \mathbf{S}_{\text{Max}}$. Intuitively, if the game is at state $s \in \mathbf{S}_{\text{Max}}$, h is the sequence of executed actions and player Max plays using strategy σ then Max will play action $a \in \mathbf{A}(s)$ with probability $\sigma(h, s)(a)$.

The strategy σ is *memoryless* (or stationary) if the past history is not taken into account, i.e. if $\sigma(h, s) = \sigma(\mathbf{1}, s)$ for each finite history $h \in \mathbf{A}^*$, where $\mathbf{1}$ is the empty history.

The strategy σ is *deterministic* (or pure) if for each finite history h and each state $s \in \mathbf{S}_{\text{Max}}$ the support of $\sigma(h, s)$ consists of one action.

Thus a memoryless deterministic strategy σ for player Max is just a mapping $\sigma : \mathbf{S}_{\text{Max}} \to \mathbf{A}$ such that, for each state $s \in \mathbf{S}_{\text{Max}}$, $\sigma(s) \in \mathbf{A}(s)$. Intuitively, $\sigma(s)$ is the action that player Max plays each time s is visited.

Strategies of player Min (general, memoryless, deterministic) are defined mutatis mutandis.

Our basic probability space associated with a given arena is the space $(\mathbf{A}^\omega, \mathcal{B}(\mathbf{A}^\omega))$ of infinite histories (infinite action sequences) equipped with the Borel σ-algebra. The basic stochastic process associated with each game is the process $\{A_i; i \in \mathbb{N}\}$ with values in $\mathbf{A}$, where A_i is the action taken at stage i. Another process of interest is the auxiliary stochastic process $\{S_i; i \in \mathbb{N}\}$ with values in $\mathbf{S}$ defined as $S_i = \text{source} \circ A_i$, i.e. S_i gives the source of the ith action (or equivalently the state at stage i).

Fixing strategies σ and τ of players Max and Min and an initial probability distribution over states $\mu \in \mathcal{D}(\mathbf{S})$ there exists a unique probability measure $\mathrm{P}_\mu^{\sigma,\tau}$ on $(\mathbf{A}^\omega, \mathcal{B}(\mathbf{A}^\omega))$ satisfying the following conditions:

$$\mathrm{P}_\mu^{\sigma,\tau}\{S_1 = s\} = \mu(s), \tag{2}$$

i.e. the initial state probability is given by μ,

$$\mathrm{P}_\mu^{\sigma,\tau}\{A_{n+1} = a_{n+1} | A_1 = a_1, \ldots, A_n = a_n, S_{n+1} = s_{n+1}\} = \begin{cases} \sigma(a_1 \ldots a_n, s_{n+1})(a_{n+1}) & \text{if player}(s_{n+1}) = \text{Max}, \\ \tau(a_1 \ldots a_n, s_{n+1})(a_{n+1}) & \text{if player}(s_{n+1}) = \text{Min}, \end{cases} \tag{3}$$

i.e. if $a_1 \ldots a_n$ is the current history and s_{n+1} the current state then the probability distribution over actions taken on stage $n+1$ is dictated by the strategy of the player controlling s_{n+1},

$$\mathrm{P}_\mu^{\sigma,\tau}\{S_{n+1} = s_{n+1} | A_1 = a_1, \ldots, A_n = a_n\} = \delta(a_n, s_{n+1}), \tag{4}$$

i.e. the state on stage $n+1$ depends only on the action executed at stage n.

2.3 Payoff mappings

After an infinite play player Max receives a payoff from player Min. The players have opposite goals, Max wishes to maximize the payoff while player Min wants to minimize the payoff.

A payoff function is a Borel measurable mapping

$$u : \mathbf{A}^\omega \to (-\infty, \infty]$$

from infinite histories to real numbers extended with plus infinity. To avoid integrability problems we assume that u is bounded from below, i.e. there exists $K \in \mathbb{R}$ such that $u(h) \geq K$ for all $h \in \mathbf{A}^\omega$, and we note by $\mathcal{M}_b$ the class of such payoff functions.

A game is a couple $\Gamma = (\mathcal{A}, u)$ made of an arena and a payoff function $u \in \mathcal{M}_b$.

Let us recall that the *tail σ-algebra* relative to the sequence $\{A_i; i \in \mathbb{N}\}$ of r.v. is the σ-algebra $\bigcap_{n=1}^\infty \sigma(A_i; i \geq n)$, where $\sigma(A_i; i \geq n)$ is the σ-algebra generated by random variables $\{A_i; i \geq n\}$. Thus a payoff function u is measurable relative to the tail σ-algebra if and only if u is measurable relative to $(\mathbf{A}^\omega, \mathcal{B}(\mathbf{A}^\omega))$ and u does not depend on initial finite histories, i.e. $u(a_1 a_2 \ldots) = u(a_2 \ldots)$ for each history $h = a_1 a_2 \ldots \in \mathbf{A}^\omega$. We note by $\mathcal{T}_b$ the class of all tail measurable mappings belonging to $\mathcal{M}_b$.

2.3.1 Mean-payoff games

A reward function is a function $r : \mathbf{A} \to \mathbb{R}$. Given a reward function r the payoff of a mean-payoff game is calculated as follows:

$$u(a_1 a_2 a_3 \ldots) = \limsup \frac{1}{n} \sum_{i=1}^n r(a_i).$$

Since for a given arena the set $\mathbf{A}$ of actions is finite the payoff of mean-payoff games belongs to $\mathcal{T}_b$.

2.3.2 Parity games

In parity games we assume that there is a priority mapping $\alpha : \mathbf{A} \to \mathbb{N}$ and the payoff is calculated as

$$u(a_1 a_2 a_3 \ldots) = (\liminf \alpha(a_i)) \mod 2.$$

Again this payoff mapping belongs to $\mathcal{T}_b$.

2.4 Priority mean-payoff games

In priority mean-payoff games we combine priorities and rewards. There are several forms of such games [5, 4, 3] but the most general one is defined as follows. We have three mappings $\alpha : \mathbf{A} \to \mathbb{N}$, $w : \mathbf{A} \to \mathbb{R}_+$ and $r : \mathbf{A} \to \mathbb{R}$ assigning to each state a non-negative integer priority, a positive real weight and a real reward respectively.

The payoff is calculated in the following way. For each infinite sequence $x = a_1 a_2 a_3 \ldots$ of actions we extract the subsequence $a_{i_1} a_{i_2} a_{i_3} \ldots$ composed of all actions with priority c where c is the minimal priority such that the set $\{i \mid \alpha(a_i) = c\}$ is infinite. Then the payoff for x is calculated as

$$u(x) = \limsup \frac{\sum_{k=1}^{n} w(a_{i_k}) r(a_{i_k})}{\sum_{k=1}^{n} w(a_{i_k})}$$

i.e. this is a weighted mean-payoff but calculated only over actions with the minimal priority visited infinitely often.

The games with such payoff contain as special cases parity games as well as mean-payoff games.

2.5 Optimal strategies

Given an initial state distribution μ and strategies σ and τ of Max and Min the expected value of the payoff u under $\mathrm{P}_\mu^{\sigma,\tau}$ is denoted $\mathrm{E}_\mu^{\sigma,\tau}[u]$.

If μ is such that $\mu(s) = 1$ for some state s then to simplify the notation the corresponding probability and expectation are noted $\mathrm{P}_s^{\sigma,\tau}$ and $\mathrm{E}_s^{\sigma,\tau}$.

Given a game $(\mathcal{A}, u)$ strategies $\sigma^\sharp$ and $\tau^\sharp$ of players Max and Min are said to be *optimal* if for each state s there exists a value $\mathrm{val}(s) \in \mathbb{R}$ (the value of s) such that

$$\mathrm{E}_s^{\sigma^\sharp,\tau}[u] \geq \mathrm{val}(s) \geq \mathrm{E}_s^{\sigma,\tau^\sharp}[u]$$

for all strategies σ and τ of players Max and Min.

Martin's theorem [9] guarantees that every state has a value. However it does not guarantee the existence of optimal strategies.

3 Playing without players – 0-player games

Let $\mathcal{A}$ be an arena such that each state has only one available action. Then each player has only one possible trivial strategy consisting in choosing at each state the unique available action. Since the players have no decision to take we can as well forget them, once the game starts the actions can be executed automatically.

The resulting process $\{S_i; i \in \mathbb{N}\}$ is a (homogeneous) Markov process with states $\mathbf{S}$ and with one step transition probabilities $p : \mathbf{S} \times \mathbf{S} \to [0,1]$ such that, for all states $x, x' \in \mathbf{S}$, $p(x,x') = \delta(a_x, x')$, where a_x is the unique action available at x. Then we have $\mathrm{P}(S_n = x_n | S_1 = x_1, \ldots, S_{n-1} = x_{n-1}) = p(x_{n-1}, x_n)$ for all n.

Since we have a natural one to one correspondence between actions and states not only the process $\{A_i; i \in \mathbb{N}\}$ determines $\{S_i; i \in \mathbb{N}\}$ but also, conversely, $\{S_i; i \in \mathbb{N}\}$ determines $\{A_i; i \in \mathbb{N}\}$.

Let us recall some basic notions from the theory of Markov chains [8].

A state s of a Markov chain is said to be *transient* if the probability to return to s (for the chain starting at s) is strictly smaller than 1.

A set C of states of a Markov chain is *closed* if, for each $s \in C$ and each $t \notin C$, $p(s,t) = 0$.

A set C of states of a Markov chain is *irreducible* if for any states $s, t \in C$ there is a positive probability to enter t for a chain starting at s and vice versa.

The set of states of each Markov chain can be decomposed as $T \cup C_1 \cup \ldots \cup C_k$, where T are transient states and C_i are closed irreducible sets.

A Markov chain containing no transient states and one closed irreducible set is called irreducible.

Each finite state Markov chain enters almost surely, after a finite number steps, some closed irreducible component. Thus if the payoff is tail measurable then it is determined by its value in each such component.

As a rather straightforward consequence of the Kolmogorov $0-1$ law we obtain

▶ **Theorem 1.** *Let $u \in \mathcal{T}_b$ be a tail measurable payoff.*

Then for each irreducible Markov process with a finite state set $\mathbf{S}$ and action set $\mathbf{A}$ there exists a constant c such that, $\mathrm{P}_\mu\{w \in \mathbf{A}^\omega \mid u(w) = c\} = \mathrm{P}_\mu\{u = c\} = 1$, where P_μ is the measure on $(\mathbf{A}^\omega, \mathcal{B}(\mathbf{A}^\omega))$ induced by the Markov process with the initial state distribution μ.

Clearly Theorem 1 implies that a tail measurable payoff is almost surely constant in each closed irreducible component of a finite Markov chain.

4 Multi-armed bandits

A multi-armed bandit is just a finite sequence of Markov chains (or equivalently 0-player games) $\mathbb{B} = (\mathcal{B}_1, \ldots, \mathcal{B}_n)$. Each $\mathcal{B}_i$ is an arm of $\mathbb{B}$. We assume that each arm $\mathcal{B}_i$ is in some state s_i, thus the state of $\mathbb{B}$ is the vector $(s_1, \ldots, s_n)$, where s_i is the state of the ith arm.

Player Max plays an infinite game on $\mathbb{B}$. Let $(s_1, \ldots, s_n)$ be the state of $\mathbb{B}$. Player Max chooses one of the arms i, the nature executes the unique action available at s_i, $\mathcal{B}_i$ enters a new state s_i', and $(s_1, \ldots, s_i', \ldots, s_n)$ becomes the new global state of $\mathbb{B}$.

A payoff mapping is defined on the set of infinite sequence of actions of $\mathbb{B}$ and, as usually, player Max wants to maximize the expected payoff. A multi-armed bandit game is a pair $(\mathbb{B}, u)$ consisting of a multi-armed bandit and a payoff mapping. Thus a multi-armed bandit game is just a special type of a one-player stochastic game.

We say the multi-armed bandit is irreducible if each $\mathcal{B}_i$ is an irreducible Markov chain.

▶ **Definition 2.** A strategy of player Max in an irreducible multi-armed bandit is said to be trivial if at each step Max chooses the same arm i.

Note that each trivial strategy is deterministic and memoryless but the triviality condition is stronger, if $\mathbb{B}$ is composed of more than one Markov chain then there are many deterministic memoryless strategies that are not trivial in the sense of Definition 2.

It is easy to see that a multi-armed irreducible bandit with the mean payoff or with the parity payoff has optimal trivial strategies. The same holds for priority mean-payoff.

In general the question if there exists a trivial optimal strategy for a multi-armed bandit game is easier to handle than the question if there exists an optimal memoryless deterministic strategy for the corresponding one-player stochastic game thus it is interesting to note that the last problem can be reduced to the former one.

5 Optimal strategies for one-player games

In this section we consider general one-player games with a tail measurable payoff. We assume that all states of $\mathcal{A} = (\mathbf{S}, \mathbf{A}, \mathrm{source}, \delta, \mathrm{player})$ are controlled by the same player, without a loss of generality we assume that this is player Max.

We call such an arena a one-player arena. A one-player game (or a Markov decision process) is a game on a one-player arena.

We examine the question when player Max has an optimal deterministic memoryless strategy for a given one-player game $(\mathcal{A}, u)$ with $u \in \mathcal{T}_b$.

It turns out that this question can be reduced to the problem of the existence of trivial optimal strategies for some related irreducible multi-armed bandit games.

An arena $\mathcal{A}_\sharp = (\mathbf{S}_\sharp, \mathbf{A}_\sharp, \mathrm{source}_\sharp, \delta_\sharp, \mathrm{player}_\sharp)$ is a subarena of $\mathcal{A} = (\mathbf{S}, \mathbf{A}, \mathrm{source}, \delta, \mathrm{player})$ if $\mathcal{A}_\sharp$ is an arena obtained from $\mathcal{A}$ by removing some states and actions. Note that the requirement that $\mathcal{A}_\sharp$ be an arena means that each state of $\mathcal{A}_\sharp$ retains at least one available action.

We say that a multi-armed bandit $\mathbb{B} = (\mathcal{B}_1, \dots, \mathcal{B}_n)$ is embeddable into an arena $\mathcal{A}$ if each $\mathcal{B}_i$ is a subarena of $\mathcal{A}$. Note that we allow the same chain to be used several times in $\mathbb{B}$, i.e. the chains $\mathcal{B}_i$ and $\mathcal{B}_j$ can be equal even if $i \neq j$. This implies that each finite arena $\mathcal{A}$ has an infinite number of embeddable multi-armed bandits.

The following theorem reduces the question about the existence of optimal memoryless deterministic strategies in one-player games to a question about optimal trivial strategies in related multi-armed bandit games:

▶ **Theorem 3.** *Let $(\mathcal{A}, u)$ be a one-player game with $u \in \mathcal{T}_b$.*

If for each irreducible multi-armed bandit $\mathbb{B}$ embeddable in $\mathcal{A}$ there exists an optimal trivial strategy in the game $(\mathbb{B}, u)$ then player Max *has an optimal memoryless deterministic strategy in $(\mathcal{A}, u)$.*

In particular we can immediately deduce that one-player parity games, mean-payoff games and priority mean-payoff games have optimal memoryless deterministic strategies.

However the real value of Theorem 3 is not in recovering old results, I hope that it will prove to be useful for establishing the existence of optimal memoryless deterministic strategies for new games.

6 From one-player games to two-player games

Let $\mathcal{A} = (\mathbf{S}, \mathbf{A}, \mathrm{source}, \delta, \mathrm{player})$ and $\mathcal{A}_\sharp = (\mathbf{S}_\sharp, \mathbf{A}_\sharp, \mathrm{source}_\sharp, \delta_\sharp, \mathrm{player}_\sharp)$ be two arenas. A morphism from $\mathcal{A}_\sharp$ to $\mathcal{A}$ is a pair (f, g) of mappings $f : \mathbf{S}_\sharp \to \mathbf{S}$ and $g : \mathbf{A}_\sharp \to \mathbf{A}$ such that

- for each $s_\sharp \in \mathbf{S}_\sharp$, $\mathrm{player}_\sharp(s_\sharp) = \mathrm{player}(f(s_\sharp))$, i.e. f preserves the controlling player,
- for each $a_\sharp \in \mathbf{A}_\sharp$, $f(\mathrm{source}_\sharp(a_\sharp)) = \mathrm{source}(g(a_\sharp))$, i.e. the source of each action is preserved,
- for each $s_\sharp \in \mathbf{S}_\sharp$, for $a_\sharp, b_\sharp \in \mathbf{S}_\sharp(s_\sharp)$, if $a_\sharp \neq b_\sharp$ then $g(a_\sharp) \neq g(b_\sharp)$, i.e. g is locally surjective (but actions with different sources can be mapped to the same action),

- (f, g) preserves (positive) transition probabilities, for all $s_\sharp \in \mathbf{S}_\sharp$ and $a_\sharp \in \mathbf{A}_\sharp$, if $\delta_\sharp(a_\sharp, s_\sharp) > 0$ then $\delta(g(a_\sharp), f(s_\sharp)) = \delta_\sharp(a_\sharp, s_\sharp)$.

The degree of the morphism (f, g) is defined as $\max_{s\in\mathbf{S}} |f^{-1}(s)|$.

Let $(\mathcal{A}, u)$ be a game and (f, g) a morphism from an arena $\mathcal{A}_\sharp$ to $\mathcal{A}$. The lifting of u is the payoff mapping $u_\sharp : (\mathbf{A}_\sharp)^\omega \to (-\infty, \infty]$ such that, for $w = a_1a_2a_3 \ldots \in (\mathbf{A}_\sharp)^\omega$, $u_\sharp(w) = u(g(w))$, where $g(w) = g(a_1)g(a_2)g(a_3)\ldots$. The game $(\mathcal{A}_\sharp, u_\sharp)$ will be called the lifting of $(\mathcal{A}, u)$ through the morphism (f, g).

In this section we adopt a slightly extended notion of a one-player arena. We say that $\mathcal{A}$ is a one-player arena controlled by player Max if for each state s controlled by player Min the set $\mathbf{A}(s)$ of available actions contains only one element.

Since for states s with one available action it does not matter who controls s this modified notion of a one-player arena is essentially equivalent to the one used in the previous section.

The following is an enhanced version of the main result of [6]:

▶ **Theorem 4.** *Let $\Gamma = (\mathcal{A}, u)$ be a two-player game. Suppose that for each morphism (f, g) of degree at most 2 from a one-player arena $\mathcal{A}_\sharp$ to $\mathcal{A}$ the player controlling $\mathcal{A}_\sharp$ has an optimal deterministic memoryless strategy in the corresponding lifted one-player game $\Gamma_\sharp = (\mathcal{A}_\sharp, u_\sharp)$. Then both players have optimal deterministic memoryless strategies in Γ.*

Note that for each arena $\mathcal{A}$ there is only a finite number of morphisms of degree at most 2 into $\mathcal{A}$. Thus Theorem 4 states that to establish the existence of optimal memoryless deterministic strategies in a two-player game it suffices to examine a finite number of one-player games.

Note also that a lifting of a mean-payoff game is again a mean-payoff game, similarly a lifting of a parity game is a parity game, and a lifting of a priority mean-payoff game is a priority mean-payoff game thus Theorem 4 allows to deduce that two-player versions of these games have optimal deterministic memoryless strategies for both players. Again these results are not new and the true value of Theorem 4 resides rather in potential applications to new games.

7 Final remarks

There is a large body of literature devoted to multi-armed bandits but it concerns mainly bandits with discounted payoff. The result announced in Theorem 3 relating Markov decision processes to multi-armed bandits seems to be new. Another sufficient condition for the existence of optimal memoryless deterministic strategies in one-player games with a tail measurable payoff is due to H. Gimbert [2]:

▶ **Theorem 5** (H. Gimbert)**.** *Let u be a tail-measurable payoff. Suppose that for all infinite words (histories) $w, w_1, w_2 \in \mathbf{A}^\omega$ such that w is a shuffle of w_1 and w_2, u satisfies the following inequality*

$$u(w) \leq \max\{u(w_1), u(w_2)\}.$$

Then finite state Markov decision processes (one-player games controlled by Max*) with payoff u have optimal deterministic memoryless strategies.*

In practice it is easier to verify the condition of Theorem 5 than the one stated in Theorem 3. However the condition of Theorem 3 seems to be more robust since there exist one-player games where we can prove the existence of optimal memoryless deterministic strategies by means of Theorem 3 but not by Theorem 5 (at least not directly).

It is an open problem how to extend Theorem 3 to payoffs which are measurable but not tail measurable.

References

1 J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer, 1997.

2 H. Gimbert. Pure stationary optimal strategies in Markov decision processes. In *STACS 2007, 24th Annual Symposium on Theoretical Aspects of Computer Science*, volume 4393 of *LNCS*, pages 200–211. Springer, 2007.

3 H. Gimbert and W. Zielonka. Limits of multi-discounted Markov decision processes. In *22th Annual IEEE Symposium on Logics in Computer Science, LICS 2007*, pages 89–98. IEEE Computer Society, 2007.

4 H. Gimbert and W. Zielonka. Perfect information stochastic priority games. In *ICALP 2007*, volume 4596 of *LNCS*, pages 850–861. Springer, 2007.

5 H. Gimbert and W. Zielonka. Blackwell-optimal strategies in priority mean-payoff games. In *GandALF 2010, First International Symposium on Games, Automata, Logics and Formal Verification*, volume 25 of *Electronic Proceedings in Theoretical Computer Science*, pages 7–21, 2010.

6 H. Gimbert and W. Zielonka. Pure nad stationary optimal strategies in perfect-information stochastic games. Technical report, HAL 00438359, 2010. Available as `http://hal.archives-ouvertes.fr/hal-00438359/en`.

7 E. Grädel, W. Thomas, and T. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*. Springer, 2002.

8 G. Grimmett and D. Stirzaker. *Probability and Random Processes*. Oxford University Press, 2001.

9 D.A. Martin. The determinacy of Blackwell games. *Journal of Symbolic Logic*, 63(4):1565–1581, 1998.

Better Algorithms for Satisfiability Problems for Formulas of Bounded Rank-width*

Robert Ganian, Petr Hliněný, and Jan Obdržálek

Faculty of Informatics, Masaryk University
Botanická 68a, Brno, Czech Republic
{xganian1,hlineny,obdrzalek}@fi.muni.cz

Abstract

We provide a parameterized polynomial algorithm for the propositional model counting problem #SAT, the runtime of which is single-exponential in the rank-width of a formula. Previously, analogous algorithms have been known – e.g. [Fischer, Makowsky, and Ravve] – with a single-exponential dependency on the clique-width of a formula. Our algorithm thus presents an exponential runtime improvement (since clique-width reaches up to exponentially higher values than rank-width), and can be of practical interest for small values of rank-width. We also provide an algorithm for the Max-SAT problem along the same lines.

Keywords and phrases propositional model counting; satisfiability; rank-width; clique-width; parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.73

1 Introduction

The satisfiability problem for Boolean formulas in conjunctive normal form (known as SAT) has been of great practical and theoretical interest for decades. It is known to be NP-complete, even though many instances are practically solvable using the various SAT-solvers. We focus on two well-known generalizations of this problem, namely #SAT and Max-SAT. In #SAT– otherwise known as the *propositional model counting* problem – the goal is to compute the number of satisfying truth assignments for an input formula ϕ, whereas in Max-SAT we ask for the maximum number of simultaneously satisfiable clauses of ϕ. It is known that computing #SAT is #P-hard [21] and that Max-SAT is already NP-hard to approximate within some constant [1].

In light of these hardness results, we may ask what happens if we restrict ourselves to some subclass of inputs. The parameterized algorithmics approach is suitable in such a case. Let k be some parameter associated with the input instance. Such a decision problem is said to be *fixed-parameter tractable (FPT)* if it is solvable in time $\mathcal{O}(n^p \cdot f(k))$ for some constant p and a computable function f. So the running time is polynomial in n, the size of the input, but can be e.g. exponential in the parameter k. Obviously the specific form of f plays an important role in practical applicability of any such algorithm – while FPT algorithms with single-exponential f can be feasible for non-trivial values of the parameter, a double-exponential f would make the algorithm impractical for almost all values of k.

But what are suitable parameters for satisfiability problems? In the particular case of Max-SAT, one can consider the desired number of satisfied or unsatisfied clauses as a

* This research has been supported by the Czech bilateral research grant GA 201/09/J021 and by the Institute for Theoretical Computer Science ITI, project 1M0545.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 73–83

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

parameter of the input, such as in [4, 19], respectively. However such approach is not at all suitable for #SAT which is our prime interest in this paper.

Another approach used for instance by Fischer, Makowsky and Ravve [7] represents the formula ϕ as a formula graph F_ϕ (nodes of which are the clauses and variables of ϕ, see Definition 6), and exploits the fact that for graphs there are many known (and intensively studied) so-called *width parameters*. In [7] the authors presented FPT algorithms for the #SAT problem in the case of two well known width parameters – *tree-width* and *clique-width*. A similar idea was used by Georgiou and Papakonstantinou [10] also for the Max-SAT problem and by Samer and Szeider [20] for #SAT.

The latter algorithms work by dynamic programming on tree-like decompositions related to the width parameters (tree-decompositions and clique-decompositions – often called k-expressions – in the cases above). However, there is the separate issue of the complexity of computing the width of the formula graph and its decomposition. In the case of tree-width this can be done in FPT [2]. For the much more general clique-width (every graph of bounded tree-width also has bounded clique-width, while the converse does not hold) there exist no such algorithms and we rely on approximations or an oracle. In [20] the authors made the following statement on this issue:

> A single-exponential algorithm (for #SAT) is due to Fisher, Makowsky, and Ravve [7]. However, both algorithms rely on clique-width approximation algorithms. The known polynomial-time algorithms for that purpose admit an exponential approximation error [13] and are of limited practical value.

The exponential approximation error mentioned in this statement results by bounding the clique-width by a another, fairly new, width parameter called *rank-width* (Definition 2). Rank-width is bounded if and only if clique-width is bounded, but its value can be exponentially lower than that of clique-width (Theorem 3 a,b). And since clique-width generalizes tree-width, so does rank-width (Theorem 3 c). Moreover, for rank-width we can efficiently compute the related decomposition (Theorem 4), which is in stark contrast to the case for clique-width. Therefore an algorithm which is linear in the formula size and *single-exponential in its rank-width* challenges the claim quoted above, and can be of real practical value. In this paper we present such algorithms for the problems #SAT and Max-SAT (Theorems 9 and 17). Precisely we prove the following two results:

▶ **Theorem 1.** *Both the* #SAT *and* Max-SAT *problems have FPT algorithms running in time*

$$\mathcal{O}(t^3 \cdot 2^{3t(t+1)/2} \cdot |\phi|),$$

provided a rank-decomposition of the input instance (CNF formula) ϕ of width t is given on the input. If, on the other hand, no rank-decomposition is given along with the input ϕ, then the runtime estimate is

$$\mathcal{O}(2^{\Theta(t^2)} \cdot |\phi|^3)$$

where t is the rank-width of the input instance (CNF formula) ϕ.

We refer to further Theorems 11 and 9, 17 for details and the proofs.

Note that our results present an *actual exponential runtime improvement* in the parameter over any algorithm utilizing the clique-width measure, including aforementioned [7]. This is since any parameterized algorithm $\mathcal{A}$ for a SAT problem has to depend at least exponentially on the clique-width of a formula (unless the Exponential Time Hypothesis fails). Then, considering typical instances ϕ as from Proposition 8 b, such an algorithm $\mathcal{A}$ runs in time

which is double-exponential in the rank-width of ϕ even if $\mathcal{A}$ is given an optimal clique-width expression on the input.

As for potential practical usefulness of Theorem 1, note that there are no "huge hidden constants" in the $\mathcal{O}$-notation. One may also ask whether there are any interesting classes of graphs of low rank-width. The answer is a resounding YES, since already for $t = 1$ we obtain the very rich class of distance-hereditary graphs. Rank-width indeed is a very general graph width measure.

The approach we use to prove both parts of Theorem 1 quite naturally extends the elaborated new algebraic methods of designing parameterized algorithms for graphs of bounded rank-width, e.g. [6, 3, 8], to the area of SAT problems. Yet, this is not a trivial extension—we remark that a straightforward translation of the algorithm of [7] from clique-width expressions to rank-decompositions (which is easily possible) would result just in a double-exponential runtime dependency on the rank-width.

The rest of the paper is organized as follows: In Section 2 we present the rank-width measure and some related technical considerations. This is applied to signed graphs of SAT formulas. Section 3 then presents our FPT algorithm for the #SAT problem (Theorem 9 and Algorithm 16), and Section 4 the similar algorithm for Max-SAT (Theorem 17). We conclude with some related observations.

2 Definitions

2.1 Rank-width

The usual way of defining rank-width [18] is via the branch-width of the cut-rank function (Definition 2). A set function $f : 2^M \to \mathbb{Z}$ is *symmetric* if $f(X) = f(M \setminus X)$ for all $X \subseteq M$. A tree is *subcubic* if all its nodes have degree at most 3. For a symmetric function $f : 2^M \to \mathbb{Z}$ on a finite ground set M, the branch-width of f is defined as follows:

A *branch-decomposition* of f is a pair (T, μ) of a subcubic tree T and a bijective function $\mu : M \to \{t : t \text{ is a leaf of } T\}$. For an edge e of T, the connected components of $T \setminus e$ induce a bipartition (X, Y) of the set of leaves of T. The *width* of an edge e of a branch-decomposition (T, μ) is $f(\mu^{-1}(X))$. The *width* of (T, μ) is the maximum width over all edges of T. The *branch-width* of f is the minimum of the width of all branch-decompositions of f.

▶ **Definition 2. (Rank-width [18])** For a simple undirected graph G and $U, W \subseteq V(G)$, let $\boldsymbol{A}_G[U, W]$ be the matrix defined over the two-element field GF(2) as follows: the entry $a_{u,w}$, $u \in U$ and $w \in W$, of $\boldsymbol{A}_G[U, W]$ is 1 if and only if uw is an edge of G. The *cut-rank* function $\rho_G(U) = \rho_G(W)$ then equals the rank of $\boldsymbol{A}_G[U, W]$ over GF(2) where $W = V(G) \setminus U$. A *rank-decomposition* (see Figure 1) and *rank-width* of a graph G is the branch-decomposition and branch-width of the cut-rank function ρ_G of G on $M = V(G)$, respectively.

As already mentioned in the introduction, rank-width is closely related to clique-width and more general than better known tree-width. Indeed:

▶ **Theorem 3.** *Let G be a simple graph, and* $\mathrm{tw}(G)$, $\mathrm{bw}(G)$, $\mathrm{cwd}(G)$ *and* $\mathrm{rwd}(G)$ *denote in this order the tree-width, branch-width, clique-width, and rank-width of G. Then*

a) [18] $\mathrm{rwd}(G) \leq \mathrm{cwd}(G) \leq 2^{\mathrm{rwd}(G)+1} - 1$,
b) [5] the clique-width $\mathrm{cwd}(G)$ *can reach up to* $2^{\mathrm{rwd}(G)/2-1}$,
c) [17] $\mathrm{rwd}(G) \leq \mathrm{bw}(G) \leq \mathrm{tw}(G) + 1$,
d) [folklore] $\mathrm{tw}(G)$ *cannot be bounded from above by a function of* $\mathrm{rwd}(G)$*, e.g. the complete graphs have rank-width* 1 *while their tree-width is unbounded,*

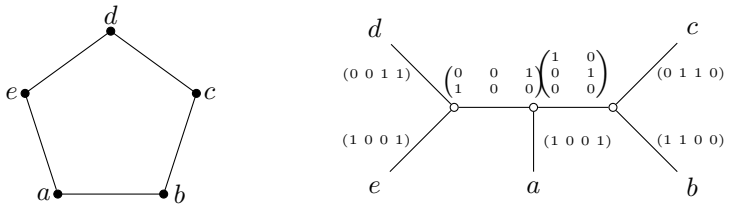


Figure 1 A rank-decomposition of the graph cycle C_5, showing the matrices involved in evaluation of its cut-rank function on the edges of the decomposition.

e) [15] $\mathrm{rwd}(G) = 1$ *if and only if* G *is a distance-hereditary graph.*

Although rank-width and clique-width are "tied together" by Theorem 3 a, one of the crucial advantages of rank-width is its parameterized tractability (note that it is not known how to efficiently test $\mathrm{cwd}(G) \leq k$ for $k > 3$):

▶ **Theorem 4.** [13] *There is an FPT algorithm that, for a fixed parameter* t *and a given graph* G*, either finds a rank-decomposition of* G *of width at most* t *or confirms that the rank-width of* G *is more than* t *in time* $O(|V(G)|^3)$*.*

With regard to our interest in precise runtime dependency on the parameter rank-width one should ask how the FPT algorithm of Theorem 4 depends on t. This issue is not at all addressed in [13], but a second look at the algorithm reveals a worst-case triple-exponential dependency on t (this is the currently best upper bounded on the number of forbidden minors for the matroids of branch-width t). One can do better while sacrificing the exact value of rank-width, such as using [18] or an improvement of the second algorithm of [16]:

▶ **Theorem 5.** [16] *There is an algorithm that, for a fixed parameter* t *and a given graph* G*, either finds a rank-decomposition of* G *of width at most* $3t$ *or confirms that the rank-width of* G *is more than* t *in time* $O(2^{\Theta(t^2)} \cdot |V(G)|^3)$*.*

2.2 Signed graphs of CNF formulas

There are several methods for converting formulas to graphs, the two most common and perhaps most natural approaches utilizing directed graphs (as seen e.g. in [20]) or so-called *signed graphs* (e.g. [7, 20, 12]). We employ the latter approach for technical reasons, although all the results also transfer straightforwardly to the former one. A signed graph is a graph G with two edge sets $E^+(G)$ and $E^-(G)$. We refer to its respective positive and negative subgraphs as to G^+ and G^-. Notice that G^+ and G^- are edge-disjoint and $G = G^+ \cup G^-$.

▶ **Definition 6.** The signed graph F_ϕ of a CNF formula ϕ is defined as follows:
- $V(F_\phi) = W \cup C$ where W is the set of variables of ϕ and C is the set of clauses of ϕ.
- For $w \in W$ and $c \in C$, it is $wc \in E^+(F_\phi)$ iff the literal 'w' occurs in c.
- For $w \in W$ and $c \in C$, it is $wc \in E^-(F_\phi)$ iff the literal '$\neg w$' occurs in c.

Since signed graphs have two distinct edge sets, the definition of rank-width needs to be modified to reflect this. It should be noted that simply using two separate, independent decompositions would not work – the bottom-up dynamic programming algorithm we are going to use will need information from both edge sets at every node to work properly. Instead, one may define, analogically to Definition 2, the *signed rank-width* of a signed graph G as the branch-width of the signed cut-rank function $\rho^{\pm}_G(U) = \rho_{G^+}(U) + \rho_{G^-}(U)$.

▶ **Definition 7. (Rank-width of formulas)** The *(signed) rank-width* $\mathrm{rwd}(\phi)$ of a CNF formula ϕ is the signed rank-width of the signed formula graph F_ϕ.

We remark that, although our signed rank-width is essentially equivalent to an existing concept of bi-rank-width of directed graphs as introduced by Kanté [14] (in the bipartite case, at least), the latter concept is not widely known and its introduction in the context of CNF formulas would bring only additional technical complications.

In our paper we propose signed rank-width as a way of measuring complexity of formulas that fares significantly better than previously considered signed clique-width of F_ϕ (e.g. [7]). *Signed clique-width* is the natural extension of clique-width having two separate operators for creating the 'plus' and the 'minus' edges. The main advantage of using rank-width as a parameter instead of clique-width comes from the following claim:

▶ **Proposition 8.** *Let ϕ be an arbitrary CNF formula and let $\mathrm{cwd}(\phi)$ denote the signed clique-width of F_ϕ. Then the following are true*

a) $\mathrm{rwd}(\phi) \leq 2\,\mathrm{cwd}(\phi)$,
b) there exist instances ϕ such that $\mathrm{cwd}(\phi) \geq 2^{\mathrm{rwd}(\phi)/4-1}$.

Proof. a) Assume a signed k-expression tree S for F_ϕ where $k = \mathrm{cwd}(\phi)$. Clearly, S gives ordinary k-expression trees for each of F^+_ϕ, F^-_ϕ. Now, analogically to Theorem 3 a [18], the rank-decompositions of F^+_ϕ and F^-_ϕ with the same underlying tree as S have widths $\leq k$ each, and hence $\mathrm{rwd}(\phi) \leq k + k = 2\,\mathrm{cwd}(\phi)$.

b) We define ϕ such that $F^+_\phi = G$ where, cf. Theorem 3 b, $\mathrm{cwd}(G) \geq 2^{\mathrm{rwd}(G)/2-1}$, and F^-_ϕ is arbitrary such that its rank-decomposition inherited from that of G has width $\leq \mathrm{rwd}(G)$. Then $\mathrm{rwd}(\phi) \leq 2\,\mathrm{rwd}(G)$ and the claim follows since $\mathrm{cwd}(\phi) \geq \mathrm{cwd}(G)$. ◀

3 Algorithm for Propositional Model Counting #SAT

This section proves our main result – the #SAT part of Theorem 1:

▶ **Theorem 9.** *Given a CNF formula ϕ and a (t^+, t^-)-labeling parse tree of the signed formula graph F_ϕ, there is an algorithm that counts the number of satisfying assignments of ϕ in time*

$$\mathcal{O}(t^3 \cdot 2^{3t(t+1)/2} \cdot |\phi|) \qquad \text{where } t = \max(t^+, t^-).$$

Note that the statement speaks about so called labeling parse trees (in place of rank-decompositions) which is a technical algebraic concept [6] suited for easier handling of rank-decompositions – this concept is briefly introduced in Section 3.1 to follow.

Our algorithm (see Algorithm 16) proving Theorem 9 applies the dynamic programming paradigm on the parse tree of the formula graph F_ϕ (constructed by Theorem 11). This is, on one hand, a standard approach utilized also by Fischer, Makowsky and Ravve [7]. On the other hand, however, in comparison to [7] we achieve an exponential runtime speedup in terms of rank-width, exploiting the nice linear-algebraic properties of rank-decompositions and their parse trees – see Section 3.2.

3.1 Parse trees for rank-decompositions

First of all we remark that, unlike for tree-width or clique-width, a bare rank-decomposition is not suitable for immediate design of algorithms. The remedy is provided by the notion of labeling parse trees [6, 8] which, informally saying, "enrich" a rank-decomposition with additional information captured in labelings of the vertices.

A *t-labeling* of a graph is a mapping $lab : V(G) \to 2^{[t]}$ where $[t] = \{1, 2, \ldots, t\}$ is the set of *labels*. Having a graph G with an associated t-labeling lab, we refer to the pair $\bar{G} = (G, lab)$ as to a *t-labeled graph*. A canonical operation associated with t-labeled graphs $\bar{G}_1 = (G_1, lab^1)$ and $\bar{G}_2 = (G_2, lab^2)$ is the *labeling join* $\bar{G}_1 \otimes \bar{G}_2$, defined on the disjoint union of G_1 and G_2 by adding all edges (u, v) such that $|lab^1(u) \cap lab^2(v)|$ is odd, where $u \in V(G_1), v \in V(G_2)$ (the resulting graph is unlabeled).

The labeling concept is similar to the definition of clique-width where every graph vertex carries one label (while here we allow a set of labels). It is actually more powerful to view a t-labeling of G equivalently as a mapping $V(G) \to \mathrm{GF}(2)^t$ into the *binary vector space* of dimension t, where GF(2) is the two-element finite field. (Then an edge $\{u, v\}$ is added in $\bar{G}_1 \otimes \bar{G}_2$ if and only if $lab^1(u) \cdot lab^2(v) = 1$ as a scalar product of vectors.)

As shown first by Courcelle and Kanté in [6], a rank-decomposition of width t can be equivalently characterized by a t-labeling parse tree (thereafter called an algebraic expression for rank-width). A *t-labeling parse tree* T for the t-labeled graph $\bar{G}$ is a finite rooted ordered subcubic tree such that

- the leaves of T form the vertex set of $\bar{G}$, and
- for each internal node z of T, the subtree $T_z \subseteq T$ rooted at z builds up a t-labeled graph $\bar{G}_z$ which (up to relabeling) equals the subgraph of $\bar{G}$ induced on the leaves of T_z.

To keep this paper simple and accessible, we skip a (rather long) formal definition of the algebraic operations taking place in the internal nodes of a t-labeling parse tree. Instead, we summarize their properties which will be used in our algorithm:

▶ **Property 10.** [8] *Let a t-labeled graph $\bar{G}$ have a t-labeling parse tree T. For every internal node z of T with a son (left or right) x, the following holds.*

a) The labels of the vertices of $V(G_x)$ in $\bar{G}_z$ are obtained from the corresponding labels in $\bar{G}_x$ by a linear transformation (relabeling) over the binary vector space $\mathrm{GF}(2)^t$.
b) For each vertex $v \in V(G_z)$, the $\bar{G}$-neighbours of v in the set $V(G) \setminus V(G_z)$ depend only on the label of v in $\bar{G}_z$. More precisely, there exists a t-labeled graph $\bar{H}$ such that unlabeled H is equal to the subgraph of G induced on $V(G) \setminus V(G_z)$, and $G = \bar{G}_z \otimes \bar{H}$.

In the context of signed graphs we, moreover, introduce the following two straightforward extensions:

- A signed graph G with associated pair of labelings $lab^+ : V(G) \to 2^{[t^+]}$ and $lab^- : V(G) \to 2^{[t^-]}$ is called a (t^+, t^-)-*labeled graph* $\widetilde{G} = (G, lab^+, lab^-)$. We use $\widetilde{G}^+$ as a shorthand for the t^+-labeled graph (G^+, lab^+), and $\widetilde{G}^-$ is defined analogically. The scope of the join operation $\otimes$ is then extended in a natural way: $\widetilde{G}_1 \otimes \widetilde{G}_2 = \big(\widetilde{G}_1^+ \otimes \widetilde{G}_2^+\big) \cup \big(\widetilde{G}_1^- \otimes \widetilde{G}_2^-\big)$.
- A (t^+, t^-)-*labeling parse tree* $T = (T^+, T^-)$ of a signed graph G is a pair (T^+, T^-) of labeling parse trees T^+ and T^- such that T^+ (T^-) is a t^+-labeling (t^--labeling) parse tree generating G^+ (G^-), and the underlying rooted trees of T^+ and T^- are identical.

The underlying idea is to separately handle the labelings for the positive and negative subgraphs of signed G, while keeping the same parse tree structure for both.

Finally, we shall use the following extension of Theorem 5 which makes restricted use of "partitioned" rank-width of vertex-partitioned graphs.

▶ **Theorem 11.** *There is an algorithm that, for a fixed parameter t and a given CNF formula ϕ, in time $O(2^{\Theta(t^2)} \cdot |\phi|^3)$ either finds a (t^+, t^-)-labeling parse tree for the formula graph F_ϕ where $t^+ \leq 3t$ and $t^- \leq 3t$, or confirms that the signed rank-width of ϕ is more than t.*

3.2 Recording partial assignments of a formula

The core of every dynamic programming algorithm is the specification of information which is then (exhaustively) processed on the input parse tree. Here we heavily apply the basic calculus of linear algebra (which is indeed natural in view of the definition of rank-width).

We say that labeling ℓ is *orthogonal* to a set of labelings X if ℓ has an even intersection with every element of X (i.e. the scalar product of the labeling vectors is 0 over GF(2)). Remember that for t-labeled graphs, in order for two vertices to become adjacent by the join operation $\otimes$, their labelings need to have an odd intersection, i.e. to be non-orthogonal. The power of orthogonality comes from the following rather trivial claim appearing already in [3, 8]:

▶ **Lemma 12.** [3, 8] *Assume t-labeled graphs $\bar{G}$ and $\bar{H}$, and arbitrary $X \subseteq V(\bar{G})$ and $y \in V(\bar{H})$. In the join graph $\bar{G} \otimes \bar{H}$, the vertex y is adjacent to some vertex in X if and only if the vector subspace spanned by the $\bar{G}$-labelings of the vertices of X is not orthogonal to the $\bar{H}$-labeling vector of y in* $\mathrm{GF}(2)^t$.

In view of Lemma 12, the following result will be useful in deriving the complexity of our algorithm.

▶ **Lemma 13.** [11] (cf. [8, Proposition 6.1]) *The number $S(t)$ of subspaces of the binary vector space* $\mathrm{GF}(2)^t$ *satisfies $S(t) \leq 2^{t(t+1)/4}$ for all $t \geq 12$.*

In the course of computation of our algorithm we need to remember some local information about all satisfying assignments for ϕ. The information to be remembered for each such assignment will be formally described in Definition 14. Before that, we would like to informally introduce its key idea of recording an "expectation" (when processing the parse tree of the input) – in addition to the information recorded about a partial solution processed so far, we also keep what is expected from a complementary partial solution coming from the unprocessed part of the input (cf. Definition 14. II).

The idea behind the algorithm is that the amount of information one has to remember about a partial solution shrinks a lot if one "knows" what the complete solution will look like. Such saving sometimes largely exceeds the cost of keeping an exhaustive list of all possible future shapes of complete solutions. This is also our case: we may exhaustively preprocess the values of some variables in advance.

The idea of using an "expectation" to speed up a dynamic programming algorithm on a rank-decomposition has first appeared in Bui-Xuan, Telle and Vatshelle [3] in relation to solving the dominating set on graphs of bounded rank-width. This concept has been subsequently formalized and generalized by the authors in [8] (in the so called PCE scheme formalism). Furthermore, it has also been shown [8, Proposition 5.1] that use of the "expectation" concept is unavoidable to achieve speed up for the dominating set problem which shares, in a sense, a similar background with our situation.

By Definition 6 the signed graph F_ϕ of a formula ϕ has a vertex set $V(F_\phi) = W \cup C$ where W is the set of variables and C is the set of clauses of ϕ. An *assignment* is then a mapping $\nu : W \to \{0, 1\}$. We speak about a *partial assignment* if ν is a partial mapping from a specific subset of W (a situation that is typical in dynamic processing). For every partial assignment we then define:

▶ **Definition 14.** Consider an arbitrary (t^+, t^-)-labeling $\widetilde{F}_1 = (F_1, lab^+, lab^-)$ of a signed subgraph $F_1 \subseteq F_\phi$, and any partial assignment $\nu_1 : V(F_1) \cap W \to \{0, 1\}$. We say that ν_1 is an *assignment of shape* $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1$ if

I. Σ^+ is the subspace of $\mathrm{GF}(2)^t$ generated by the set of labeling vectors $lab^+(\nu_1^{-1}(1))$ and Σ^- is the subspace of $\mathrm{GF}(2)^t$ generated by $lab^-(\nu_1^{-1}(0))$, and
II. Π^+, Π^- (the *expectation part* of the shape) are subspaces of $\mathrm{GF}(2)^t$ such that, for every clause $c \in V(F_1) \cap C$, at least one of the following is true
   - c is adjacent to some vertex from $\nu_1^{-1}(1)$ in F_1^+ or to one from $\nu_1^{-1}(0)$ in F_1^-, or
   - the label vector $lab^+(c)$ is not orthogonal to Π^+ or $lab^-(c)$ is not orthogonal to Π^- (cf. Lemma 12).

Very informally saying, I. states which true literals in F_1 (w.r.t. ν_1) are available to satisfy clauses of F_ϕ, and II. stipulates that every clause in F_1 is already satisfied by a literal in F_1 or is expected to be satisfied by some literal in $F_\phi \setminus V(F_1)$. Note that one partial assignment ν_1 could be of several distinct shapes, which differ just in the expectation part, i.e. in Π^+, Π^-. This is true even for complete assignments. Moreover, there is no requirement on Π^+ and Π^- to have an empty intersection with Σ^+, Σ^- and each other.

From the definition, considering Property 10 b and Lemma 12, one easily gets:

▶ **Proposition 15.** *We consider a CNF formula ϕ with the variable set W, and any assignment $\nu : W \to \{0,1\}$. Assume $\widetilde{F}_1, \widetilde{F}_2$ are (t^+,t^-)-labeled graphs such that $F_\phi = \widetilde{F}_1 \otimes \widetilde{F}_2$, and let ν_1, ν_2 denote the restrictions of ν to $\widetilde{F}_1, \widetilde{F}_2$.*

a) The assignment ν is satisfying for ϕ if, and only if, there exist subspaces Σ^+, Σ^-, Π^+, Π^- of $\mathrm{GF}(2)^t$ such that ν_1 is of shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1$ and ν_2 is of shape $(\Pi^+, \Pi^-, \Sigma^+, \Sigma^-)$ in $\widetilde{F}_2$.
b) If, in $\widetilde{F}_1$, ν_1 is of shape $(\Sigma_0^+, \Sigma_0^-, \Pi_0^+, \Pi_0^-)$ and, at the same time, ν_1 is of shape $(\Sigma_1^+, \Sigma_1^-, \Pi_1^+, \Pi_1^-)$, then $\Sigma_0^+ = \Sigma_1^+$ and $\Sigma_0^- = \Sigma_1^-$.
c) The assignment ν_1 is satisfying for ϕ_1 – the subformula of ϕ represented by F_1 if, and only if, ν_1 is of shape $(\Sigma^+, \Sigma^-, \emptyset, \emptyset)$ for some subspaces Σ^+, Σ^-.

◀

3.3 The dynamic processing algorithm

We now return to the proof of our Theorem 9.

▶ **Algorithm 16.** (Theorem 9) *Given is a CNF formula ϕ and a signed (t^+,t^-)-labeling parse tree T_ϕ of the formula graph F_ϕ. Our task is to compute the total number of satisfying assignments of ϕ:*

At every node z such that the subtree of T_ϕ rooted at z parses a (t^+,t^-)-labeled graph $\widetilde{F}_z$, we create an integer-valued array $Table_z$ indexed by all the quadruples of subspaces of $\mathrm{GF}(2)^t$, where $t = \max(t^+,t^-)$. The value of the entry $Table_z[\Sigma^+, \Sigma^-, \Pi^+, \Pi^-]$ shall be equal to the total number of variable assignments in the subformula of ϕ corresponding to $F_z \subseteq F_\phi$ that are of the shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_z$ (cf. Definition 14).

The computation is (briefly, details in [9]) as follows.

1. We initialize all entries of $Table_z$ for $z \in V(T_\phi)$ to 0.
2. We process all nodes of T_ϕ in the leaves-to-root order as follows.
   a) At a clause leaf c of T_ϕ, we set $Table_c[\emptyset, \emptyset, \Pi^+, \Pi^-] = 1$ (there is just one, empty, possible partial assignment in $\widetilde{F}_c$) for all pairs of expectation subspaces Π^+, Π^- that would satisfy this clause c.
   b) At a variable leaf ℓ of T_ϕ, we record one partial variable assignment with $\ell = 1$ and one with $\ell = 0$, both for all possible expectation pairs Π^+, Π^- (this is since there is no clause in $\widetilde{F}_\ell$, and so any expectation would work).

c) Consider an internal node z of T_ϕ, with the left son x and the right son y such that $Table_x$ and $Table_y$ have already been computed. We loop exhaustively over all "compatible" triples of indices to $Table_x$, $Table_y$, and $Table_z$ as follows.
   - We say that the indices in $Table_x[\Sigma_x^+, \Sigma_x^-, \Pi_x^+, \Pi_x^-]$, $Table_y[\Sigma_y^+, \Sigma_y^-, \Pi_y^+, \Pi_y^-]$, and $Table_z[\Sigma_z^+, \Sigma_z^-, \Pi_z^+, \Pi_z^-]$ are *compatible* if, e.g., the space Σ_z^+ spans the union of relabeled (cf. Property 10 a) spaces Σ_x^+, Σ_y^+, and the expectation space Π_x^+ is spanned by the (again relabeled accordingly) expectation space Π_z^+ and the space Σ_y^+ (which records the true literals of the corresponding $Table_y$ entry). Analogical conditions ought to be true for Σ_z^-, and $\Pi_x^-, \Pi_y^+, \Pi_y^-$.
   - Whenever such a compatible triple of table indices is found, we add up the product $Table_x[\Sigma_x^+, \Sigma_x^-, \Pi_x^+, \Pi_x^-] \cdot Table_y[\Sigma_y^+, \Sigma_y^-, \Pi_y^+, \Pi_y^-]$ to the entry $Table_z[\Sigma_z^+, \Sigma_z^-, \Pi_z^+, \Pi_z^-]$.

3. For the root r of T_ϕ, we sum up all the entries $Table_r[\Sigma^+, \Sigma^-, \emptyset, \emptyset]$ where Σ^+, Σ^- are arbitrary subspaces of $\mathrm{GF}(2)^t$. This is the resulting number of satisfying assignments of ϕ according to Proposition 15 c.

Correctness of Algorithm 16 routinely follows from Definition 14 of assignment shapes, Properties 10 of labeling parse trees, and Proposition 15. Our key novel contribution, on the other hand, is the runtime analysis of this algorithm in which we employ subspace orthogonality and the notion of *expectation* outlined in Section 3.2.

Runtime of Algorithm 16 is dominated by the calls to step (2c). Although we say that we exhaustively loop over all 12-tuples of subspaces of $\mathrm{GF}(2)^t$ there, actually half of the subspaces are determined by the others using linear algebra. Hence one call to this step takes time $\mathcal{O}\big(t^3 \cdot S(t)^6\big)$ where $S(t)$ bounds the number of subspaces as in Lemma 13. Altogether our Algorithm 16 takes time

$$\mathcal{O}\big(|V(T_\phi)| \cdot t^3 \cdot S(t)^6\big) = \mathcal{O}\big(|V(T_\phi)| \cdot t^3 \cdot 2^{6t(t+1)/4}\big) = \mathcal{O}\big(|\phi| \cdot t^3 \cdot 2^{3t(t+1)/2}\big).$$

4 Algorithm for the Max-SAT Problem

The same ideas as presented in Section 3 lead also to a parameterized algorithm for the Max-SAT optimization problem which asks for the maximum number of satisfied clauses in a CNF formula. We briefly describe this extension, though we have to admit that the importance of the Max-SAT algorithm on graphs of bounded rank-width is not as high as that of #SAT. The reason for lower applicability is that for "sparse" formula graphs (i.e. those not containing large bipartite cliques) their rank-width is bounded iff their tree-width is bounded, while for dense formula graphs the satisfiability problem is easier in general.

▶ **Theorem 17.** *There is an algorithm that, given a CNF formula ϕ and a (t^+, t^-)-labeling parse tree of the formula graph F_ϕ, solves the* Max-SAT *optimization problem of ϕ in time $\mathcal{O}(t^3 \cdot 2^{3t(t+1)/2} \cdot |\phi|)$ where $t = \max(t^+, t^-)$.*

In order to formulate this algorithm, we extend Definition 14 as follows. Recall $V(F_\phi) = W \cup C$ where W are the variables and C are the clauses of ϕ.

▶ **Definition 18.** Consider a (t^+, t^-)-labeling $\widetilde{F}_1 = (F_1, lab^+, lab^-)$ of a signed subgraph $F_1 \subseteq F_\phi$, and a partial assignment $\nu_1 : V(F_1) \cap W \to \{0, 1\}$. We say that ν_1 is an *assignment of defective shape* $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1$ if there exists a set $C_0 \subseteq C \cap V(F_1)$ such that ν_1 is of shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$ in $\widetilde{F}_1 - C_0$. The value (the *defect*) of ν_1 with respect to this defective shape is the minimum cardinality of such C_0.

Informally, the defect equals the number of clauses in F_1 which are unsatisfied there and not expected to be satisfied in a complete assignment in F_ϕ.

▶ **Algorithm 19.** (Theorem 17) *Given is a CNF formula ϕ and a signed (t^+, t^-)-labeling parse tree T_ϕ of the formula graph F_ϕ. Our task is to compute the maximum number of satisfied clauses over all variable assignments of ϕ. A brief sketch follows, while the details can be found in [9]:*

We process the parse tree T_ϕ of F_ϕ similarly to Algorithm 16, but this time the value of the entry $Table_z[\Sigma^+, \Sigma^-, \Pi^+, \Pi^-]$ shall be equal to the minimum defect over all partial assignments in $\widetilde{F}_z$ that are of defective shape $(\Sigma^+, \Sigma^-, \Pi^+, \Pi^-)$. Within the framework of Algorithm 16 we, in an internal node z of T_ϕ with the sons x, y, determine the defect of any partial assignment in $\widetilde{F}_z$ (with respect to a particular shape, cf. Definition 18) as the sum of the defects of the corresponding (inherited) partial assignments in $\widetilde{F}_x$ and $\widetilde{F}_y$. The implementation is similar to the former.

At the end of processing, in the root r of T_ϕ, we find the minimum m over all the entries $Table_r[\Sigma^+, \Sigma^-, \emptyset, \emptyset]$ where Σ^+, Σ^- are arbitrary subspaces of $\mathrm{GF}(2)^t$. An optimal solution to Max-SAT of ϕ then has $|C| - m$ satisfied clauses.

5 Conclusions

We have presented new FPT algorithms for the #SAT and Max-SAT problems on formulas of bounded rank-width. Our algorithms are single-exponential in rank-width and linear in the size of the formula (cubic if a rank-decomposition has to be computed first), and they do not involve any "large hidden constants". This is a significant improvement over previous results, for several reasons. In the case of tree-width this follows from the fact that rank-width is much less restrictive than tree-width. If a graph has bounded tree-width it also has bounded rank-width, but there are classes of graphs with arbitrarily high tree-width and small rank-width (e.g. cliques, complete bipartite graphs, or distance hereditary graphs).

As for clique-width (which is bounded iff rank-width is bounded), we have obtained two significant improvements over the existing algorithms such as [7]. Firstly, rank-width can be exponentially smaller than clique-width, and therefore we obtain an exponential speed-up over the existing algorithms in the worst case. Secondly, there is an FPT algorithm for computing the rank-width of a graph (and the associated rank-decomposition) exactly, or a faster one providing a factor-3 approximation, whereas in the case of clique-width we have to rely on an approximation by an exponential function of rank-width.

Finally, our paper shows that many of the recent ideas and tricks of parameterized algorithm design on graphs of bounded rank-width smoothly translate to certain SAT-related problem instances, a fact which may also provide new inspiration for related research.

References

1 S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45:70–122, 1998.

2 H. Bodlaender. Treewidth: Algorithmic techniques and results. In *MFCS'97*, volume 1295 of *LNCS*, pages 19–36. Springer, 1997.

3 B.-M. Bui-Xuan, J. A. Telle, and M. Vatshelle. H-join decomposable graphs and algorithms with runtime single exponential in rankwidth. *Discrete Appl. Math.*, 158(7):809–819, 2010.

4 J. Chen and I. A. Kanj. Improved exact algorithms for Max-Sat. *Discrete Appl. Math.*, 142(1-3):17–27, 2004.

5 D. Corneil and U. Rotics. On the relationship between cliquewidth and treewidth. *SIAM J. Comput.*, 34(4):825–847, 2005.

6 B. Courcelle and M. Kanté. Graph operations characterizing rank-width. *Discrete Appl. Math.*, 157(4):627–640, 2009.

7 E. Fischer, J.A. Makowsky, and E. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discrete Appl. Math.*, 156:511–529, 2008.

8 R. Ganian and P. Hliněný. On parse trees and Myhill–Nerode–type tools for handling graphs of bounded rank-width. *Discrete Appl. Math.*, 158:851–867, 2010.

9 R. Ganian, P. Hliněný, and J. Obdržálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. arXiv:1006.5621v1 [cs.DM], June 2010.

10 K. Georgiou and P.A. Papakonstantinou. Complexity and algorithms for well-structured k-SAT instances. In *SAT'08*, pages 105–118, 2008.

11 J. Goldman and G.-C. Rota. The number of subspaces of a vector space. In W.T. Tutte, editor, *Recent Progress in Combinatorics*, pages 75–83. Academic Press, 1969.

12 G. Gutin, E. J. Kim, S. Szeider, and A. Yeo. Solving MAX-2-SAT above a tight lower bound. arXiv:0907.4573, July 2009.

13 P. Hliněný and S. Oum. Finding branch-decomposition and rank-decomposition. *SIAM J. Comput.*, 38:1012–1032, 2008.

14 M. Kanté. The rank-width of directed graphs. arXiv:0709.1433v3, March 2008.

15 S. Oum. Rank-width and vertex-minors. *J. Comb. Theory Ser. B*, 95(1):79–100, 2005.

16 S. Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5(1):1–20, 2008.

17 S. Oum. Rank-width is less than or equal to branch-width. *J. Graph Theory*, 57(3):239–244, 2008.

18 S. Oum and P. D. Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

19 I. Razgon and B. O'Sullivan. Almost 2-SAT is fixed-parameter tractable. *J. Comput. System Sci.*, 75(8):435–450, 2009.

20 M. Samer and S. Szeider. Algorithms for propositional model counting. *J. Discrete Alg.*, 8:50–64, 2010.

21 L. G. Valiant. The complexity of computing the permanent. *Theoret. Comput. Sci.*, 8(2):189–201, 1979.

Satisfiability of Acyclic and Almost Acyclic CNF Formulas*

Sebastian Ordyniak[1], Daniel Paulusma[2], and Stefan Szeider[1]

1 **Institute of Information Systems**
Vienna University of Technology, A-1040 Vienna, Austria
sebastian.ordyniak@tuwien.ac.at, stefan@szeider.net
2 **School of Engineering and Computing Sciences**
Durham University, Durham, DH1 3LE England
daniel.paulusma@durham.ac.uk

Abstract

We study the propositional satisfiability problem (SAT) on classes of CNF formulas (formulas in Conjunctive Normal Form) that obey certain structural restrictions in terms of their hypergraph structure, by associating to a CNF formula the hypergraph obtained by ignoring negations and considering clauses as hyperedges on variables. We show that satisfiability of CNF formulas with so-called "β-acyclic hypergraphs" can be decided in polynomial time.

We also study the parameterized complexity of SAT for "almost" β-acyclic instances, using as parameter the formula's distance from being β-acyclic. As distance we use the size of smallest strong backdoor sets and the β-hypertree width. As a by-product we obtain the W[1]-hardness of SAT parameterized by the (undirected) clique-width of the incidence graph, which disproves a conjecture by Fischer, Makowsky, and Ravve (*Discr. Appl. Math.* 156, 2008).

Keywords and phrases Satisfiability, chordal bipartite graphs, β-acyclic hypergraphs, backdoor sets, parameterized complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.84

1 Introduction

We study the propositional satisfiability problem (SAT) on classes of CNF formulas (formulas in Conjunctive Normal Form) that obey certain structural restrictions in terms of their hypergraph structure, associating to a CNF formula the hypergraph obtained from the formula by ignoring negations and considering clauses as hyperedges on variables.

Many otherwise hard problems become easy if restricted to acyclic instances. Hence it is natural to ask if SAT becomes polynomial-time tractable for CNF formulas with acyclic hypergraphs. However, in contrast to graphs, there are several notions of acyclicity for hypergraphs: α-acyclicity, β-acyclicity, γ-acyclicity, and Berge acyclicity, as described and discussed by Fagin [5]. We will provide definitions for the notions of acyclicity that are relevant to this paper in Section 2. It is known that the various notions of acyclicity are strictly ordered with respect to their generality, i.e., we have

$$\alpha\text{-}\textsc{Acyc} \supsetneq \beta\text{-}\textsc{Acyc} \supsetneq \gamma\text{-}\textsc{Acyc} \supsetneq \text{Berge-}\textsc{Acyc} \tag{1}$$

where X-$\textsc{Acyc}$ denotes the class of X-acyclic hypergraphs. Let X-$\textsc{Acyc}$-$\textsc{Sat}$ denote the propositional satisfiability problem restricted to X-acyclic CNF formulas (i.e., CNF formulas

* Ordyniak and Szeider's research was funded by the ERC (COMPLEX REASON, 239962). Paulusma's research was funded by the EPSRC (EP/G043434/1).

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 84–95

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

whose associated hypergraph is X-acyclic). It is not difficult to see that α-Acyc-Sat is NP-complete (see [19]), and that Berge-Acyc-Sat is solvable in polynomial time. For the latter, observe that if a CNF formula F is Berge-acyclic, then its incidence graph has treewidth 1, thus whether F is satisfiable can be decided in linear time [6, 19].

It is natural to ask where in the chain (1) the exact boundary between NP-completeness and polynomial-time tractability lies. In Section 3 we will answer this question and show that β-Acyc-Sat (and thus also γ-Acyc-Sat) can be solved in polynomial time. We establish this result by combining techniques from structural graph theory (a connection between β-acyclic hypergraphs and chordal bipartite graphs) with a fundamental technique for satisfiability solving (Davis-Putnam resolution).

In Sections 4 and 5 we will explore possibilities to gradually generalize the class of β-acyclic CNF formulas. We will study the parameterized complexity of deciding the satisfiability of formulas parameterized by their "distance" from the class of β-acyclic CNF formulas, with respect to two distance measures.

The first distance measure is based on the notion of *strong backdoor sets*: For a CNF formula F we define its "distance to β-acyclicity" as the size k of a smallest set B of variables such that for each partial truth assignment to B, the reduct of F under the assignment is β-acyclic (such a set B is a strong backdoor set). If we know B, then clearly deciding the satisfiability of F reduces to deciding the satisfiability of at most 2^k β-acyclic CNF formulas, and is thus fixed-parameter tractable with respect to k. We show, however, that finding such a set B of size k (if it exists) is W[2]-hard, thus unlikely fixed-parameter tractable for parameter k which limits the algorithmic usefulness of this distance measure.

The second distance measure we consider is the *β-hypertree width*, a hypergraph invariant introduced by Gottlob and Pichler [11]. The classes of hypergraphs of β-hypertree width $k = 1, 2, 3, \ldots$ form an infinite chain of proper inclusions. Hypergraphs of β-hypertree width 1 are exactly the β-acyclic hypergraphs. Thus β-hypertree width is also a way to define a "distance to β-acyclicity." The complexity of determining the β-hypertree width of a hypergraph is open [11]. However, we show that even if we are given the CNF formula together with a hypertree decomposition of width k, deciding the satisfiability of F parameterized by k is W[1]-hard. As a side effect, we obtain from this result that SAT is also W[1]-hard when parameterized by the *clique-width* (of the undirected incidence graph) of the CNF formula. This disproves a conjecture by Fischer, Makowsky, and Ravve [6].

2 Preliminaries

We assume an infinite supply of propositional *variables*. A *literal* is a variable x or a negated variable $\overline{x}$; if $y = \overline{x}$ is a literal, then we write $\overline{y} = x$. For a set S of literals we put $\overline{S} = \{\, \overline{x} \mid x \in S \,\}$; S is *tautological* if $S \cap \overline{S} \neq \emptyset$. A *clause* is a finite non-tautological set of literals. A finite set of clauses is a *CNF formula* (or *formula*, for short). A variable x *occurs* in a clause C if $x \in C \cup \overline{C}$; $\mathsf{var}(C)$ denotes the set of variables which occur in C. For a formula F we put $\mathsf{var}(F) = \bigcup_{C \in F} \mathsf{var}(C)$. Let F be a formula and $X \subseteq \mathsf{var}(F)$. We denote by F_X the set of clauses of F in which some variable of X occurs; i.e., $F_X := \{\, C \in F \mid \mathsf{var}(C) \cap X \neq \emptyset \,\}$.

A *truth assignment* is a mapping $\tau : X \to \{\, 0, 1 \,\}$ defined on some set X of variables; we write $\mathsf{var}(\tau) = X$. For $x \in \mathsf{var}(\tau)$ we define $\tau(\overline{x}) = 1 - \tau(x)$. For a truth assignment τ and a formula F, we define $F[\tau] = \{\, C \setminus \tau^{-1}(0) \mid C \in F,\ C \cap \tau^{-1}(1) = \emptyset \,\}$, i.e., $F[\tau]$ denotes the result of instantiating variables according to τ and applying the usual simplifications. A truth assignment τ *satisfies* a clause if the clause contains some literal x with $\tau(x) = 1$; τ satisfies a formula F if it satisfies all clauses of F (i.e., if $F[\tau] = \emptyset$). A formula is *satisfiable* if

it is satisfied by some truth assignment; otherwise it is *unsatisfiable*. Two formulas F and F' are *equisatisfiable* if either both are satisfiable or both are unsatisfiable. The SATISFIABILITY (SAT) problem is to test whether a given CNF formula is satisfiable.

A *hypergraph* H is a pair (V, E) where V is the set of *vertices* and E is the set of *hyperedges*, which are subsets of V. If $|e| = 2$ for all $e \in E$ then H is also called a *graph*. We say that a hypergraph $H' = (V', E')$ is a *partial hypergraph* of $H = (V, E)$ if $V' \subseteq V$ and $E' \subseteq E$. The *incidence graph* $I(H)$ of hypergraph $H = (V, E)$ is the bipartite graph where the sets V and E form the two partitions, and where $e \in E$ is incident with $v \in V$ if and only if $v \in e$. A hypergraph is *α-acyclic* if it can be reduced to the empty hypergraph by repeated application of the following rules:

1. Remove hyperedges that are empty or contained in other hyperedges.
2. Remove vertices that appear in at most one hyperedge.

A hypergraph H is *β-acyclic* if every partial hypergraph of H is α-acyclic. The *hypergraph* $H(F)$ of a formula F has vertex set $\mathsf{var}(F)$ and hyperedge set $\{\, \mathsf{var}(C) \mid C \in F \,\}$. We say that F is *α-acyclic* or *β-acyclic* if $H(F)$ is α-acyclic or β-acyclic, respectively. The *incidence graph* of F is the graph $I(F)$ with vertex set $\mathsf{var}(F) \cup F$ and edge set $\{\, Cx \mid C \in F \text{ and } x \in \mathsf{var}(C) \,\}$. The *directed incidence graph* of F is the directed graph with vertex set $\mathsf{var}(F) \cup F$ and arc set $\{\, (C, x) \mid C \in F \text{ and } x \in C \,\} \cup \{\, (x, C) \mid C \in F \text{ and } \overline{x} \in C \,\}$. We can also represent the orientation of edges by labeling them with the signs $+, -$, such that an edge between a variable x and a clause C is labeled $+$ if $x \in C$ and labeled $-$ if $\overline{x} \in C$. This gives rise to the *signed incidence graph* which carries exactly the same information as the directed incidence graph.

Note that one can make a hypergraph α-acyclic by adding a universal hyperedge that contains all vertices (the first step of the above reduction removes all other hyperedges, the second step all vertices). Using this fact, it is easy to see that SAT is NP-complete for the class of α-acyclic CNF formulas [19]. In contrast, it is well known that the satisfiability of α-acyclic instances of the CONSTRAINT SATISFACTION PROBLEM (CSP) can be decided in polynomial time [9]. Thus SAT and CSP behave differently with respect to acyclicity (representing a clause with k literals as a relational constraint requires exponential space of order $k2^k$).

2.1 Parameterized Complexity

We define the basic notions of Parameterized Complexity and refer to other sources [4, 7] for an in-depth treatment. A parameterized problem can be considered as a set of pairs (I, k), the instances, where I is the main part and k is the parameter. The parameter is usually a non-negative integer. A parameterized decision problem is *fixed-parameter tractable* if there exists a computable function f such that instances (I, k) of size n can be decided in time $f(k)n^{O(1)}$. The class FPT denotes the class of all fixed-parameter tractable decision problems.

Parameterized complexity offers a completeness theory, similar to the theory of NP-completeness, that allows the accumulation of strong theoretical evidence that some parameterized problems are not fixed-parameter tractable. This theory is based on a hierarchy of complexity classes $\mathrm{W}[1], \mathrm{W}[2], \ldots, \mathrm{XP}$. Each class contains all parameterized decision problems that can be reduced to a certain fixed parameterized decision problem under *parameterized reductions*. These are many-to-one reductions where the parameter for one problem maps into the parameter for the other. More specifically, problem L reduces to problem L' if there is a mapping R from instances of L to instances of L' such that (i) (I, k)

is a yes-instance of L if and only if $(I', k') = R(I, k)$ is a yes-instance of L', (ii) $k' = g(k)$ for a computable function g, and (iii) R can be computed in time $f(k)n^{O(1)}$ where f is a computable function and n denotes the size of (I, k). The class W[1] is considered as the parameterized analog to NP.

3 Polynomial-time SAT Decision for β-acyclic CNF Formulas

▶ **Theorem 1.** *Satisfiability of β-acyclic CNF formulas can be decided in polynomial time.*

The remainder of this section is devoted to a proof of this result. We need two known results on *chordal bipartite* graphs, which are bipartite graphs with no induced cycle on 6 vertices or more. The first one is Proposition 2. The equivalence between statement (i) and (ii) in this proposition is a result of Tarjan and Yannakakis [20]. The equivalence between statement (ii) and (iii) in Proposition 2 follows from the facts that $I(H(F))$ is obtained from $I(F)$ after removing all but one clause vertices in $I(F)$ with the same neighbors (i.e., clauses with the same set of variables in F) and that a chordal bipartite graph remains chordal bipartite under vertex deletion.

▶ **Proposition 2.** *Let F be a CNF formula. Then the following three statements are equivalent:*

(i) *$H(F)$ is β-acyclic;*
(ii) *$I(H(F))$ is chordal bipartite;*
(iii) *$I(F)$ is chordal bipartite.*

A vertex v in a graph G is *weakly simplicial* if (i) the neighborhood of v in G forms an independent set, and (ii) the neighborhoods of the neighbors of v form a chain under set inclusion. Uehara [21] showed the following (which also follows from results of Hammer, Maffray, and Preismann [12], see [17]). We call a bipartite graph *nontrivial* if it contains at least one edge.

▶ **Proposition 3** (Uehara [21], Hammer, Maffray, and Preismann [12])**.** *A graph is chordal bipartite if and only if every induced subgraph has a weakly simplicial vertex. Furthermore, a nontrivial chordal bipartite graph has a weakly simplicial vertex in each partite set.*

We also call a vertex of a hypergraph or a variable of a CNF formula weakly simplicial if the corresponding vertex in the associated incidence graph is weakly simplicial. The above result immediately gives a polynomial-time procedure for the recognition of β-acyclic hypergraphs: delete weakly simplicial vertices from the hypergraph as long as possible (clearly one can recognize a weakly simplicial vertex in polynomial time). The hypergraph is β-acyclic if and only if we can eliminate in this way all its vertices.

The notion of *Davis-Putnam Resolution* allows us to use this elimination procedure to decide the satisfiability of β-acyclic CNF formulas.

Let C_1, C_2 be clauses such that $C_1 \cap \overline{C}_2 = \{x\}$ for some literal x. The clause $C = (C_1 \cup C_2) \setminus \{x, \overline{x}\}$ is called the *resolvent* of C_1 and C_2 with respect to x. Note that by definition any two clauses have at most one resolvent.

Consider a formula F and a variable x of F. We obtain a formula F' from F by adding all possible resolvents with respect to x, and by removing all clauses in which x occurs. We say that F' is obtained from F by *Davis-Putnam Resolution* with respect to x and we write $\mathrm{DP}_x(F) = F'$. It is well known (and easy to show) that F and $\mathrm{DP}_x(F)$ are equisatisfiable.

The so-called Davis-Putnam Procedure [3] successively eliminates variables in this manner until either the empty formula or a formula which contains the empty clause is obtained. However, $\mathrm{DP}_x(F)$ contains in general more clauses than F. Hence, repeated application of Davis-Putnam Resolution to F may cause an exponential growth in the number of clauses. As a result, the Davis-Putnam Procedure has an exponential worst-case running time. The key observation for our algorithm is that if x is a weakly simplicial variable of F, then the size of $\mathrm{DP}_x(F)$ is not greater than the size of F.

▶ **Lemma 4.** *If x is a weakly simplicial variable of a CNF formula F, then $|\mathrm{DP}_x(F)| \leq |F|$.*

Proof. Let x be a weakly simplicial variable of a CNF formula F. Let $F - x := \{ C \setminus \{x, \overline{x}\} \mid C \in F \}$. We show that $\mathrm{DP}_x(F) \subseteq F - x$.

Assume $C_1, C_2 \in F$ have a resolvent C with respect to x. Consequently we have $C_1 \cap \overline{C_2} \subseteq \{x, \overline{x}\}$. Because x is weakly simplicial, $\mathsf{var}(C_1) \subseteq \mathsf{var}(C_2)$ or $\mathsf{var}(C_2) \subseteq \mathsf{var}(C_1)$. Without loss of generality, assume the former is the case. If $x \in C_1$, then we have $C_1 \cap \overline{C_2} = \{x\}$, and so $C = C_2 \setminus \{\overline{x}\} \in F - x$. Similarly, if $\overline{x} \in C_1$, then we have $C_1 \cap \overline{C_2} = \{\overline{x}\}$, and so $C = C_2 \setminus \{x\} \in F - x$. Thus indeed $\mathrm{DP}_x(F) \subseteq F - x$. From $|\mathrm{DP}_x(F)| \leq |F - x| \leq |F|$ the result now follows. ◀

Now it is easy to establish Theorem 1. We extend the above elimination procedure. Assume we are given a β-acyclic CNF formula F. Then $I(F)$ is chordal bipartite due to Proposition 2. As long as possible, we select a weakly simplicial variable x and compute $\mathrm{DP}_x(F)$. Because $I(\mathrm{DP}_x(F))$ is an induced subgraph of $I(F)$, it follows that $\mathrm{DP}_x(F)$ is again β-acyclic. Moreover, $\mathrm{DP}_x(F)$ and F are equisatisfiable, and by Lemma 4, $|\mathrm{DP}_x(F)| \leq |F|$. Hence we can repeat this elimination procedure, and by induction, we will finally be left with a CNF formula F_0 that has no variables. If $F_0 = \emptyset$ then F is satisfiable, and if $F_0 = \{\emptyset\}$ then F is unsatisfiable. Because each reduction step can be carried out in polynomial time, Theorem 1 follows.

4 Backdoor Sets

Let $\mathcal{C}$ be a class of CNF formulas. Consider a CNF formula F together with a set of variables $B \subseteq \mathsf{var}(F)$. We say that B is a *strong backdoor set* of F with respect to *base class* $\mathcal{C}$ if for all truth assignments $\tau : B \to \{0, 1\}$ we have $F[\tau] \in \mathcal{C}$. In that case we also say that B is a *strong $\mathcal{C}$-backdoor set*. For every CNF formula F and every set $B \subseteq \mathsf{var}(F)$ it holds that F is satisfiable if and only if $F[\tau]$ is satisfiable for at least one truth assignment $\tau : B \to \{0, 1\}$. Thus, if B is a strong $\mathcal{C}$-backdoor set of F, then determining whether F is satisfiable reduces to the satisfiability problem of at most $2^{|B|}$ reduced CNF formulas $F[\tau] \in \mathcal{C}$.

Now consider a base class $\mathcal{C}$ of CNF formulas for which SAT can be solved in polynomial time. Then, if we have found a strong $\mathcal{C}$-backdoor set of F of size k, deciding the satisfiability of F is fixed-parameter tractable for parameter k. Hence, the key question is whether we can find a strong backdoor set of size at most k if it exists. To study this question, we consider the following parameterized problem that is defined for every fixed base class $\mathcal{C}$.

Strong $\mathcal{C}$-Backdoor
Instance: A formula F and an integer $k > 0$.
Parameter: The integer k.
Question: Does F have a strong $\mathcal{C}$-backdoor set of size at most k?

It is known that Strong $\mathcal{C}$-Backdoor is fixed-parameter tractable for the class $\mathcal{C}$ of Horn formulas and for the class $\mathcal{C}$ of 2CNF formulas [14]. Let BAC be the class of all β-acyclic CNF formulas. Contrary to the above results, we show that Strong BAC-Backdoor is $W[2]$-hard.

▶ **Theorem 5.** *The problem* Strong BAC-Backdoor *is* W[2]*-hard.*

Proof. Let $\mathcal{S}$ be a family of finite sets $S_1, \ldots, S_m$. Then a subset $R \subseteq \bigcup_{i=1}^{m} S_i$ is called a *hitting set* of $\mathcal{S}$ if $R \cap S_i \neq \emptyset$ for $i = 1, \ldots, m$. The Hitting Set problem is defined as follows.

Hitting Set
Instance: A family $\mathcal{S}$ of finite sets $S_1, \ldots, S_m$ and an integer $k > 0$.
Parameter: The integer k.
Question: Does $\mathcal{S}$ have a hitting set of size at most k?

It is well known that Hitting Set is W[2]-complete [4]. We reduce from this problem to prove the theorem.

Let $\mathcal{S} = \{ S_1, \ldots, S_m \}$ and k be an instance of Hitting Set. We write $V(\mathcal{S}) = \bigcup_{i=1}^{m} S_i$ and construct a formula F as follows. For each $s \in V(\mathcal{S})$ we introduce a variable x_s, and we write $X = \{ x_s \mid s \in V(\mathcal{S}) \}$. For each S_i we introduce two variables h_i^1 and h_i^2. Then, for every $1 \leq i \leq m$, the formula F contains three clauses C_i, C_i^1, and C_i^2 such that:

- $C_i = \{ h_i^1, h_i^2 \}$;
- $C_i^1 = \{ h_i^1 \} \cup \{ x_s \mid s \in S_i \} \cup \{ \overline{x}_s \mid s \in V(\mathcal{S}) \setminus S_i) \}$;
- $C_i^2 = \{ h_i^2 \} \cup \{ \overline{x}_s \mid s \in V(\mathcal{S}) \}$.

We need the following claims. The first claim characterizes the induced cycles in $I(F)$ with length at least 6. We need it to prove the second claim.

Claim 1. Let D be an induced cycle in $I(F)$. Then $|V(D)| \geq 6$ if and only if $V(D) = \{h_i^1, h_i^2, x_s, C_i, C_i^1, C_i^2\}$ for some $1 \leq i \leq m$ and $s \in V(\mathcal{S})$.

We prove Claim 1 as follows. Suppose D is an induced cycle in $I(F)$ with $|V(D)| \geq 6$. By construction, D contains at least one vertex from X. Because any two vertices in X have exactly the same neighbors in $I(F)$, D contains at most one vertex from X. Hence, D contains exactly one vertex from X, let x_s be this vertex. Let C_i^j and $C_{i'}^{j'}$ be the two neighbors of x_s on D. Because x_s is the only of D that belongs to X, we find that h_i^j and $h_{i'}^{j'}$ belong to D. By our construction, C_i and $C_{i'}$ then belong to D as well. If $C_i \neq C_{i'}$, then D contains at least two vertices from X, which is not possible. Hence $C_i = C_{i'}$, as desired. The reverse implication is trivial, and Claim 1 is proven.

Claim 2. Let B be a strong BAC-backdoor set that contains variable h_i^j. Then, for any $s^* \in S_i$, the set $(B \backslash \{h_i^j\}) \cup \{x_{s^*}\}$ is a strong BAC-backdoor set.

We prove Claim 2 as follows. Let $s^* \in S_i$ and define $B' = (B \backslash \{h_i^j\}) \cup \{x_{s^*}\}$. Suppose B' is not a strong BAC-backdoor set. Then there is a truth assignment $\tau : B' \to \{0, 1\}$ with $F[\tau] \notin$ BAC. This means that $I(F[\tau])$ contains an induced cycle D with $|V(D)| \geq 6$. Because B is a strong BAC-backdoor set, h_i^j must belong to $V(D)$. We apply Claim 1 and obtain $V(D) = \{h_i^1, h_i^2, x_s, C_i, C_i^1, C_i^2\}$ for some $x_s \in X$. Suppose $\tau(x_{s^*}) = 1$. Then $C_i^1 \notin F[\tau]$. Hence $\tau(x_{s^*}) = 0$, but then $C_i^2 \notin F[\tau]$. This contradiction proves Claim 2.

We are ready to prove the claim that $\mathcal{S}$ has a hitting set of size at most k if and only if F has a strong BAC-backdoor set of size at most k.

Suppose $\mathcal{S}$ has a hitting set R of size at most k. We claim that $B = \{ x_s \mid s \in R \}$ is a strong BAC-backdoor set of F. Suppose not. Then there is a truth assignment τ with $F[\tau] \notin$ BAC. This means that $I(F[\tau])$ contains an induced cycle D with $|V(D)| \geq 6$. By Claim 1, we obtain $V(D) = \{h_i^1, h_i^2, x_s, C_i, C_i^1, C_i^2\}$ for some $1 \leq i \leq m$ and $s \in S$. Because C_i^1, C_i^2 are in $I(F[\tau])$, we find that $R \cap S_i = \emptyset$. This is not possible, because R is a hitting set of $\mathcal{S}$. Conversely, suppose F has a strong BAC-backdoor set B of size at most k. By Claim 2, we may without loss of generality assume that $B \subseteq X$. We claim that $R = \{ s \mid x_s \in B \}$ is a hitting set of $\mathcal{S}$. Suppose not. Then $R \cap S_i = \emptyset$ for some $1 \leq i \leq m$. This means that B contains no vertex from $\{ x_s \mid s \in S_i \}$. Let $\tau : B \to \{0,1\}$ be the truth assignment with $\tau(x_s) = 1$ for all $x_s \in B$. Then C_i^1 and C_i^2 are in $F[\tau]$. Let $s \in S_i$. Then the cycle D with $V(D) = \{h_i^1, h_i^2, x_s, C_i, C_i^1, C_i^2\}$ is an induced 6-vertex cycle in $I(F[\tau])$. This means that $F[\tau] \notin$ BAC, which is not possible. Hence, we have proven Theorem 5. ◀

4.1 Deletion Backdoor Sets

Let F be a formula and let $B \subseteq \mathsf{var}(F)$ be a set of variables. Then $F - B$ denotes the formula obtained from F after removing all literals x and $\overline{x}$ with $x \in B$ from the clauses in F. We call B a *deletion backdoor set* with respect to a base class $\mathcal{C}$ if $F - B \in \mathcal{C}$.

Deletion $\mathcal{C}$-backdoor sets can be seen as a relaxation of strong $\mathcal{C}$-backdoor sets if the base class $\mathcal{C}$ is *clause-induced*, i.e., if for every $F \in \mathcal{C}$ and $F' \subseteq F$, we have $F' \in \mathcal{C}$. In that case every deletion $\mathcal{C}$-backdoor set B is also a strong $\mathcal{C}$-backdoor set. This is well known [15] and can easily be seen as follows. Let $\tau : B \to \{0,1\}$ be a truth assignment. Then by definition $F[\tau] \subseteq F - B$. Because B is a deletion $\mathcal{C}$-backdoor set, $F - B \in \mathcal{C}$. Because $\mathcal{C}$ is clause-induced and $F[\tau] \subseteq F - B$, this means that $F[\tau] \in \mathcal{C}$, as required.

Now let $\mathcal{C}$ be a clause-induced base class. Let B be a smallest deletion $\mathcal{C}$-backdoor set and let B' be a smallest strong $\mathcal{C}$-backdoor set. Then, from the above, we deduce $|B'| \leq |B|$. The following example shows that $|B| - |B'|$ can be arbitrarily large.

We consider the base class BAC, which is obviously clause-induced. Let F be the formula with $\mathsf{var}(F) = \{x_1, \ldots, x_p, y_1, \ldots, y_p, z_1, \ldots, z_p\}$ for some $p \geq 1$ and clauses $C_1 = \{x_1, \ldots, x_p, y_1, \ldots, y_p\}, C_2 = \{\overline{y}_1, \ldots, \overline{y}_p, z_1, \ldots, z_p\}$ and $C_3 = \{x_1, \ldots, x_p, z_1, \ldots, z_p\}$ for some $p \geq 1$. Then $B = \{y_1\}$ is a smallest strong BAC-backdoor set. However, a smallest deletion BAC-backdoor set must contain at least p variables.

Analogously to the Strong $\mathcal{C}$-Backdoor problem we define the problem Deletion $\mathcal{C}$-Backdoor problem, where $\mathcal{C}$ is a fixed clause-induced base class.

Deletion $\mathcal{C}$-Backdoor
Instance: A formula F and an integer $k > 0$.
Parameter: The integer k.
Question: Does F have a deletion $\mathcal{C}$-backdoor set of size at most k?

Determining the parameterized complexity of Deletion BAC-Backdoor is interesting, especially in the light of our W[2]-hardness result for Strong BAC-Backdoor. In other words, is the problem of deciding whether a graph can be modified into a chordal bipartite graph by deleting at most k vertices fixed-parameter tractable in k? Marx [13] showed that the version of this problem in which the modified graph is required to be chordal instead of chordal bipartite is fixed-parameter tractable.

5 β-Hypertree Width and Clique-Width

Hypertree width is a hypergraph invariant introduced by Gottlob, Leone, and Scarcello [10]. It is defined via the notion of a *hypertree decomposition* of a hypergraph H, which is a triple $\mathcal{T} = (T, \kappa, \lambda)$ where T is a rooted tree and χ and λ are labelling functions with $\chi(t) \subseteq V(H)$ and $\lambda(t) \subseteq E(H)$, respectively, for every $t \in V(T)$, such that the following conditions hold:

1. For every $e \in E(H)$ there is a $t \in V(T)$ such that $e \subseteq \chi(t)$.
2. For every $v \in V(H)$, the set $\{ t \in V(T) \mid v \in \chi(t) \}$ induces a connected subtree of T.
3. For every $t \in V(T)$, it holds that $\chi(t) \subseteq \bigcup_{e \in \lambda(t)} e$.
4. For every $t \in V(T)$, if a vertex v occurs in some hyperedge $e \in \lambda(t)$ and if $v \in \chi(t')$ for some node t' in the subtree below t, then $v \in \chi(t)$.

The *width* of a hypertree decomposition (T, χ, λ) is $\max\{ |\lambda(t)| \mid t \in V(T) \}$. The *hypertree width*, denoted $\mathrm{hw}(H)$, of a hypergraph H is the minimum width over all its hypertree decompositions. Many NP-hard problems such as CSP or Boolean database queries can be solved in polynomial time for instances with associated hypergraphs of bounded hypertree width [9].

Gottlob and Pichler [11] defined β-hypertree width as a "hereditary variant" of hypertree width. The *β-hypertree width*, denoted $\beta\text{-hw}(H)$, of a hypergraph H is defined as the maximum hypertree width over all partial hypergraphs H' of H. Using the fact that α-acyclic hypergraphs are exactly the hypergraphs of hypertree width 1 [10], one deduces that the hypergraphs of β-hypertree width 1 are exactly the β-acyclic hypergraphs. Unfortunately, the complexity of determining the β-hypertree width of a hypergraph is not known [11]. However, we show the following. Here, a *β-hypertree decomposition* of width k of a hypergraph H is an oracle that produces for every partial hypergraph H' of H a hypertree decomposition of width at most k.

▶ **Theorem 6.** *SAT, parameterized by an upper bound k on the β-hypertree width of a CNF formula F, is* W[1]*-hard even if a β-hypertree decomposition of width k for $H(F)$ is given.*

Proof. A *clique* in a graph is a subset of vertices that are mutually adjacent. A k-partite graph is *balanced* if its k partition classes are of the same size. A *partitioned clique* of a balanced k-partite graph $G = (V_1, \ldots, V_k, E)$ is a clique K with $|K \cap V_i| = 1$ for $i = 1 \ldots, k$. We devise a parameterized reduction from the following problem, which is W[1]-complete [18].

Partitioned Clique
Instance: A balanced k-partite graph $G = (V_1, \ldots, V_k, E)$.
Parameter: The integer k.
Question: Does G have a partitioned clique?

Before we describe the reduction we introduce some auxiliary concepts. For any three variables z, x_1, x_2 let $F(z, x_1, x_2)$ denote the formula consisting of the clauses $\{z, x_1, \overline{x_2}\}$, $\{z, \overline{x_1}, x_2\}$, $\{z, \overline{x_1}, \overline{x_2}\}$, $\{\overline{z}, x_1, x_2\}$, and $\{\overline{z}, \overline{x_1}, \overline{x_2}\}$. This formula has exactly three satisfying assignments, corresponding to the vectors 000, 101, and 110. Hence each satisfying assignment sets at most one out of x_1 and x_2 to true, and if one of them is set to true, then z is set to true as well ("$z = x_1 + x_2$"). Taking several instances of this formula we can build a "selection gadget." Let $x_1, \ldots, x_m$ and $z_1, \ldots, z_{m-1}$ be variables. We define $F^{=1}(x_1, \ldots, x_m; z_1, \ldots, z_{m-1})$ as the union of $F(z_1, x_1, x_2)$, $\bigcup_{i=2}^{m-1} F(z_i, z_{i-1}, x_{i+1})$, and $\{\{z_{m-1}\}\}$. Now each satisfying assignment of this formula sets exactly one variable out of $\{x_1, \ldots, x_m\}$ to true, and, conversely, for

each $1 \leq i \leq m$ there exists a satisfying assignment that sets exactly x_i to true and all other variables from $\{x_1, \ldots, x_m\}$ to false.

Now we describe the reduction. Let $G = (V_1, \ldots, V_k)$ be a balanced k-partite graph for $k \geq 2$. We write $V_i = \{v_1^i, \ldots, v_n^i\}$. We construct a CNF formula F. As the variables of F we take the vertices of G plus new variables z_j^i for $1 \leq i \leq k$ and $1 \leq j \leq n-1$. We put $F = \bigcup_{i=0}^{k} F_i$ where the formulas F_i are defined as follows: F_0 contains for any $u \in V_i$ and $v \in V_j$ $(i \neq j)$ with $uv \notin E$ the clause $C_{u,v} = \{\overline{u}, \overline{v}\} \cup \{w \mid w \in (V_i \cup V_j) \setminus \{u, v\}\}$; for $i > 0$ we define $F_i = F^{=1}(v_1^i, \ldots, v_n^i; z_1^i, \ldots, z_{n-1}^i)$.

We will have proven Theorem 6 after showing the following two claims.

Claim 1. β-hw$(H(F)) \leq k$.

We prove Claim 1 as follows.

First we show that that β-hw$(H(F_0)) \leq k$. Let H_0' be a partial hypergraph of $H(F_0)$. Let I be the set of indices $1 \leq i \leq k$ such that some hyperedge of H_0' contains V_i. For each $i \in I$ we choose a hyperedge e_i of H_0' that contains V_i. The partial hypergraph H_0' admits a trivial hypertree decomposition (T_0, χ_0, λ_0) of width at most k with a single tree node t_0 where $\chi_0(t_0)$ contains all vertices of H_0' and $\lambda_0(t_0) = \{e_i \mid i \in I\}$. Second we observe that β-hw$(H(F_i)) = 1$ for $1 \leq i \leq k$: $H(F_i)$ is β-acyclic, and β-acyclic hypergraphs have β-hypertree width 1.

Now let H' be an arbitrarily chosen partial hypergraph of $H(F)$. For $i = 0, \ldots, k$, we let H_i' denote the (maximal) partial hypergraph of H' that is contained in $H(F_i)$. We let $\mathcal{T}_0 = (T_0, \chi_0, \lambda_0)$ be a hypertree decomposition of width at most k of H_0' as defined above. For $i = 1, \ldots, k$ we let $\mathcal{T}_i = (T_i, \chi_i, \lambda_i)$ be a hypertree decomposition of width 1 of H_i'. We combine these $k+1$ hypertree decompositions to a hypertree decomposition of width at most k for H'. We will do this by adding the decompositions $\mathcal{T}_1, \ldots, \mathcal{T}_k$ to $\mathcal{T}_0$ one by one and without increasing the width of $\mathcal{T}_0$.

Let $\mathcal{T}_i^* = (T_i^*, \chi_i^*, \lambda_i^*)$ denote the hypertree decomposition of width at most k obtained from $\mathcal{T}_0$ by adding the first i hypertree decompositions. For $i = 0$ we let $\mathcal{T}_0^* = \mathcal{T}_0$. For $i > 0$ we proceed as follows.

First we consider the case where there is a hyperedge $e \in H_0'$ with $V_{i+1} \subseteq e$. Observe that there exists a node $t \in V(T_i^*)$ with $e \subseteq \chi(t)$. We define $\mathcal{T}_{i+1}^* = (T_{i+1}^*, \chi_{i+1}^*, \lambda_{i+1}^*)$ as follows. We obtain T_{i+1}^* from the disjoint union of T_i^* and T_{i+1} by adding an edge between t and the root of T_{i+1}. As the root of T_{i+1}^* we choose the root of T_i^*. We set $\chi_{i+1}^*(t) = \chi_i^*(t)$ for every $t \in V(T_i^*)$, and $\chi_{i+1}^*(t) = \chi_{i+1}(t) \cup V_{i+1}$ for every $t \in V(T_{i+1})$; we set $\lambda_{i+1}^*(t) = \lambda_i^*(t)$ for every $t \in V(T_i^*)$, and $\lambda_{i+1}^*(t) = \lambda_{i+1}(t) \cup \{e\}$ for every $t \in V(T_{i+1})$ (hence $|\lambda_{i+1}^*(t)| \leq \max(2, k) = k$). Consequently $\mathcal{T}_{i+1}^*$ has width at most k.

It remains to consider the case where there is no hyperedge $e \in H_0'$ with $V_{i+1} \subseteq e$. We define $\mathcal{T}_{i+1}^*$ as follows. We obtain T_{i+1}^* from the disjoint union of T_i^* and T_{i+1} by adding an edge between an arbitrary node $t \in V(T_i^*)$ and the root of T_{i+1}. As the root of T_{i+1}^* we choose the root of T_i^*. We set $\chi_{i+1}^* = \chi_i^* \cup \chi_{i+1}$ and $\lambda_{i+1}^* = \lambda_i^* \cup \lambda_{i+1}$. Clearly $\mathcal{T}_{i+1}^*$ has width at most k.

Claim 2. G has a partitioned clique if and only if F is satisfiable.

To prove Claim 2 we first suppose that G has a partitioned clique K. We define a partial truth assignment $\tau : V \to \{0, 1\}$ by setting $\tau(v) = 1$ for $v \in K$, and $\tau(v) = 0$ for $v \notin K$. This partial assignment satisfies F_0, and it is easy to extend τ to a satisfying truth assignment of F. Conversely, suppose that F has a satisfying truth assignment τ. Because of the formulas F_i,

$1 \leq i \leq k$, τ sets exactly one variable $v^i_{j_i} \in V_i$ to true. Let $K = \{v^1_{j_1}, \ldots, v^k_{j_k}\}$. The clauses in F_0 ensure that $v^i_{j_i}$ and $v^{i'}_{j_{i'}}$ are adjacent in G for each pair $1 \leq i < i' \leq k$, hence K is a partitioned clique of G. This proves Claim 2. ◀

Clique-width is a graph parameter that measures in a certain sense the structural complexity of a directed or undirected graph [2]. The parameter is defined via a graph construction process where only a limited number of vertex labels are available; vertices that share the same label at a certain point of the construction process must be treated uniformly in subsequent steps. In particular, one can use the following four operations: the creation of a new vertex with label i, the vertex-disjoint union of already constructed labeled graphs, the relabeling of all vertices of label i with label j, and the insertion of all possible edges between vertices of label i and label j (either undirected or directed from label i to j). The clique-width $\mathrm{cw}(G)$ of a graph G is the smallest number k of labels that suffice to construct G by means of these four operations. Such a construction of a graph can be represented by an algebraic term called a k-expression.

The *(directed) clique-width* of a CNF formula is the clique-width of its (directed) incidence graph. The directed clique-width of a CNF formula can also be defined in terms of the signed incidence graph and is therefore sometimes called the *signed clique-width*. Observe that the clique-width of a CNF formula is always bounded by its directed clique-width. However, in general the directed clique-width can be much higher than the undirected one. It is well known that satisfiability decision is fixed-parameter tractable for the parameter directed clique-width [1, 6]. Fischer, Makowsky, and Ravve [6] developed a dynamic programming algorithm that counts the number of satisfying truth assignments in linear time for CNF formulas of bounded directed clique-width. They also conjectured that their method can be extended to work for formulas of bounded (undirected) clique-width. However, the reduction in the proof of Theorem 6 shows that this is not possible unless FPT = W[1].

▶ **Corollary 7.** *SAT, parameterized by an upper bound k on the clique-width of the incidence graph of F, is* W[1]*-hard even if a k-expression for $I(F)$ is given.*

Proof. It is easy to see that the clique-width of the incidence graph of the formula F in the proof of Theorem 6 is at most $k' = O(k)$, and a k'-expression can be found in polynomial time. Hence the result follows from the same reduction. ◀

Rank-width is a further graph parameter that has been considered for CNF formulas. This parameter was introduced by Oum and Seymour [16] for approximating the clique-width of graphs. A certain structure that certifies that a graph has rank-width at most k is called a *rank-width decomposition of width k*. Similar to clique-width, one can define the rank-width of a directed graph that takes the orientation of edges into account. The *directed* (or *signed*) *rank-width* of a CNF formula is the rank-width of its directed incidence graph. Ganian, Hliněný, and Obdržálek [8] developed an efficient dynamic programming algorithm that counts in linear time the number of satisfying assignments of a CNF formula of bounded *directed* rank-width. Because bounded undirected rank-width implies bounded undirected clique-width [16], the following is a direct consequence of Corollary 7.

▶ **Corollary 8.** *SAT, parameterized by an upper bound k on the rank-width of the incidence graph of F, is* W[1]*-hard even if a rank-decomposition of width k for $I(F)$ is given.*

6 Conclusion

We have identified a new class of CNF formulas, the class BAC of β-acyclic formulas, for which recognition and satisfiability decision are solvable in polynomial time. Furthermore, we have established hardness results for two natural strategies for gradually extending this class: extensions via strong backdoor sets and extensions via β-hypertree decompositions. The first extension is fixed-parameter intractable because the backdoor sets are hard to find; the second extension is fixed-parameter intractable because satisfiability decision remains hard even if the β-hypertree decomposition is provided. It would be interesting to know whether the satisfiability of CNF formulas of β-hypertree width bounded by an arbitrary constant k can be decided in polynomial time (of order depending on k) if a β-hypertree decomposition is provided.

References

1 Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. On the fixed parameter complexity of graph enumeration problems definable in monadic second-order logic. *Discr. Appl. Math.*, 108(1-2):23–52, 2001.

2 Bruno Courcelle, Joost Engelfriet, and Grzegorz Rozenberg. Context-free handle-rewriting hypergraph grammars. In Hartmut Ehrig, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors, *Graph-Grammars and their Application to Computer Science, 4th International Workshop, Bremen, Germany, March 5–9, 1990, Proceedings*, volume 532 of *Lecture Notes in Computer Science*, pages 253–268, 1991.

3 Martin Davis and Hillary Putnam. A computing procedure for quantification theory. *J. ACM*, 7(3):201–215, 1960.

4 Rod G. Downey and Michael R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer Verlag, New York, 1999.

5 Ronald Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *J. ACM*, 30(3):514–550, 1983.

6 Eldar Fischer, Johann A. Makowsky, and Elena V. Ravve. Counting truth assignments of formulas of bounded tree-width or clique-width. *Discr. Appl. Math.*, 156(4):511–529, 2008.

7 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*, volume XIV of *Texts in Theoretical Computer Science. An EATCS Series*. Springer Verlag, Berlin, 2006.

8 Robert Ganian, Petr Hliněný, and Jan Obdržálek. Better algorithms for satisfiability problems for formulas of bounded rank-width. Technical Report arXiv:1006.5621v1, CoRR, 2010.

9 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions: a survey. In *Mathematical foundations of computer science, 2001 (Mariánské Lázně)*, volume 2136 of *Lecture Notes in Computer Science*, pages 37–57. Springer, 2001.

10 Georg Gottlob, Nicola Leone, and Francesco Scarcello. Hypertree decompositions and tractable queries. *J. of Computer and System Sciences*, 64(3):579–627, 2002.

11 Georg Gottlob and Reinhard Pichler. Hypergraphs in model checking: acyclicity and hypertree-width versus clique-width. *SIAM J. Comput.*, 33(2):351–378, 2004.

12 Peter L. Hammer, Frederic Maffray, and Myriam Preismann. A characterization of chordal bipartite graphs. Technical report, Rutgers University, New Brunswick, NJ, 1989.

13 Dániel Marx. Chordal deletion is fixed-parameter tractable. *Algorithmica*, 57(4):747–768, 2010.

14 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Detecting backdoor sets with respect to Horn and binary clauses. In *Proceedings of SAT 2004 (Seventh International*

Conference on Theory and Applications of Satisfiability Testing, 10–13 May, 2004, Vancouver, BC, Canada), pages 96–103, 2004.

15 Naomi Nishimura, Prabhakar Ragde, and Stefan Szeider. Solving #SAT using vertex covers. *Acta Informatica*, 44(7-8):509–523, 2007.

16 Sang-il Oum and Paul Seymour. Approximating clique-width and branch-width. *J. Combin. Theory Ser. B*, 96(4):514–528, 2006.

17 Michael J. Pelsmajer, Jacent Tokazy, and Douglas B. West. New proofs for strongly chordal graphs and chordal bipartite graphs. Unpublished Manuscript, 2004.

18 Krzysztof Pietrzak. On the parameterized complexity of the fixed alphabet shortest common supersequence and longest common subsequence problems. *J. of Computer and System Sciences*, 67(4):757–771, 2003.

19 Marko Samer and Stefan Szeider. Algorithms for propositional model counting. *J. Discrete Algorithms*, 8(1):50–64, 2010.

20 Robert E. Tarjan and Mihalis Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, 13(3):566–579, 1984.

21 Ryuhei Uehara. Linear time algorithms on chordal bipartite and strongly chordal graphs. In *Automata, languages and programming*, volume 2380 of *Lecture Notes in Computer Science*, pages 993–1004. Springer, 2002.

The effect of girth on the kernelization complexity of Connected Dominating Set

Neeldhara Misra, Geevarghese Philip, Venkatesh Raman, Saket Saurabh

The Institute of Mathematical Sciences, Chennai, India.
{neeldhara,gphilip,vraman,saket}@imsc.res.in

Abstract

In the Connected Dominating Set problem we are given as input a graph G and a positive integer k, and are asked if there is a set S of at most k vertices of G such that S is a dominating set of G and the subgraph induced by S is connected. This is a basic connectivity problem that is known to be NP-complete, and it has been extensively studied using several algorithmic approaches. In this paper we study the effect of excluding short cycles, as a subgraph, on the *kernelization complexity* of Connected Dominating Set.

Kernelization algorithms are polynomial-time algorithms that take an input and a positive integer k (the *parameter*) and output an equivalent instance where the size of the new instance and the new parameter are both bounded by some function $g(k)$. The new instance is called a $g(k)$ *kernel* for the problem. If $g(k)$ is a polynomial in k then we say that the problem admits polynomial kernels. The girth of a graph G is the length of a shortest cycle in G. It turns out that Connected Dominating Set is "hard" on graphs with small cycles, and becomes progressively easier as the girth increases. More specifically, we obtain the following interesting trichotomy: Connected Dominating Set

- does not have a kernel of *any* size on graphs of girth 3 or 4 (since the problem is W[2]-hard);
- admits a $g(k)$ kernel, where $g(k)$ is $k^{\mathcal{O}(k)}$, on graphs of girth 5 or 6 but has *no* polynomial kernel (unless the Polynomial Hierarchy (PH) collapses to the third level) on these graphs;
- has a cubic ($\mathcal{O}(k^3)$) kernel on graphs of girth at least 7.

While there is a large and growing collection of parameterized complexity results available for problems on graph classes characterized by excluded *minors*, our results add to the very few known in the field for graph classes characterized by excluded *subgraphs*.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.96

1 Introduction

In the Dominating Set (DS) problem, we are given a graph G and a non-negative integer k, and the question is whether G contains a set of k vertices whose closed neighborhood contains all the vertices of G. In the connected variant Connected Dominating Set (CDS), we also demand that the subgraph induced by the dominating set be connected. DS and CDS, together with their numerous variants, are two of the most well-studied problems in algorithms and combinatorics [22]. A significant part of the algorithmic study of these NP-complete problems has focused on the design of parameterized algorithms. Informally, a *parameterization* of a problem assigns an integer k to each input instance and a parameterized problem is *fixed-parameter tractable* (FPT) if there is an algorithm that solves the problem in time $f(k) \cdot |I|^{O(1)}$, where $|I|$ is the size of the input and f is an arbitrary computable function that depends only on the parameter k. CDS is W[2]-complete on general graphs and therefore it cannot be solved by a parameterized algorithm, unless an unlikely collapse occurs in the W hierarchy (see [15, 16, 27]). However, there are interesting graph classes where FPT algorithms do exist for the Dominating Set problem. The project of widening the

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 96–107

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

horizon where such algorithms exist spawned a multitude of ideas that made DS and CDS the testbed for some of the most cutting-edge techniques of parameterized algorithm design. For example, the initial study of parameterized subexponential algorithms for DS on planar graphs [1, 10, 18] resulted in the creation of bidimensionality theory which characterizes a broad range of graph problems that admit efficient approximation schemes and/or fixed-parameter algorithms on a broad range of graphs [11, 12, 14].

Kernelization is a rapidly growing sub-area of parameterized complexity. A parameterized problem is said to admit a *polynomial kernel* if there is a polynomial time algorithm, called a *kernelization* algorithm, that reduces the input instance down to an instance with size bounded by a polynomial $p(k)$ in the parameter k, while preserving the answer. This reduced instance is called a $p(k)$ *kernel* for the problem. If $p(k) = O(k)$, then we call it a *linear kernel*. One of the first results on linear kernels is the celebrated work of Alber, Fellows, and Niedermeier on DS, on planar graphs [2]. This work spurred the interest to prove polynomial (preferably linear) kernels for other parameterized problems. The result from [2] (see also [8]) has been extended to much more general graph classes. More recently, Bodlaender et al. [5] and Fomin et al. [17] obtained algorithmic meta-kernelization results which show that a multitude of problems expressible in a certain logic (or are bidimensional) admit linear kernels on (apex) H-minor free graphs.

Most of the kernelization results mentioned above are on graph classes excluding a fixed graph as a *minor*. While there have been a lot of results obtained in the realm of parameterized algorithms on graph classes excluding some graph as a minor, there have only been a handful of such results on graph classes that are defined by excluding a fixed graph as a *subgraph*. The first result of this kind was obtained by Raman and Saurabh [29] who showed that DS and several of its variants are FPT on any class of graphs that forbids "short" cycles — cycles of length 4. This can equivalently be thought of as excluding a $K_{2,2}$, the complete bipartite graph where each part has size exactly 2. Philip et al. [28] generalized this result and showed that DS remains FPT on $K_{i,j}$-free graphs for any fixed i and j, and in fact has a polynomial kernel of size k^h where h is a constant that depends on i and j. It is a corollary of this result that the DS problem has polynomial kernels on graphs of bounded degeneracy – a class which includes graphs defined by excluding a fixed graph H as a minor. Alon and Gutner had shown previously that DS has a kernel of size $O(k^h)$ on H-minor free graphs, where the constant h depends on the excluded graph H [3, 21]. $K_{i,j}$-free graphs remain the largest class of graphs for which DS is currently known to have a polynomial sized kernel and is fixed-parameter tractable.

In this paper, we study the effect of girth on the kernelization complexity of CDS. Typically the parameterized (or other) complexity of connected variants of a problem tend to be much more than that of the problem itself. For example, Vertex Cover has a $2k$-sized vertex kernel and an efficient fixed-parameter tractable algorithm [27], and its connected variant is known not to have a polynomial sized kernel unless the Polynomial Hierarchy collapses to the third level(which is widely believed to be unlikely) [13]. Similarly, while Feedback Vertex Set has an $O^*(3.83^k)$ FPT algorithm [7], the best known FPT algorithm for its connected variant has an $O^*(c^k)$ running time [25] where c is more than 23.

The parameterized complexity of CDS has been extensively investigated, and many results are known. Thus, it is known that CDS is W[2]-hard on general graphs [15], has a linear kernel on planar, or more generally, on apex-minor-free graphs [17, 20, 24], and is FPT on graphs of bounded degeneracy [19]. CDS is also unlikely to have polynomial sized kernels on graphs of bounded degeneracy [9]. We obtain the complete kernelization complexity landscape for the CDS problem based on the girth of the problem instance. More precisely, we show that

Connected Dominating Set

1. is W[2]-hard on graphs of girth 3 or 4, and hence does not have a kernel of *any* size on these graphs unless $FPT = W[2]$;
2. has an FPT algorithm that runs in time $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$ on graphs of girth 5 or 6, and hence has a kernel of size $k^{\mathcal{O}(k)}$ on these graphs;
3. has *no* polynomial kernel (unless PH collapses to the third level) on graphs of girth 5 or 6, and,
4. has a cubic ($\mathcal{O}(k^3)$) kernel on graphs of girth at least 7.

The first result follows directly from a construction in [29], and the second and fourth results are obtained using nontrivial extensions of techniques from [29]. The main technical contribution of this paper is the third result, to obtain which we introduce an intermediate, seemingly unrelated problem (FCC), show that FCC has no polynomial kernels (unless PH collapses to the third level) using the recent kernel lower bound machinery developed by Bodlaender et al. [4], and then provide a parameter-preserving reduction [6] from FCC to CDS.

2 Preliminaries

We use $V(G)$ and to $E(G)$ denote, respectively, the vertex and edge sets of graph G. A graph H is a *subgraph* of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. The subgraph H is called an *induced subgraph* (induced by the vertex set $V(H)$) of G if $E(H) = \{\{u, v\} \in E(G) \mid u, v \in V(H)\}$. For a subset $S \subseteq V(G)$ the subgraph of G induced by S is denoted by $G[S]$, and we use $G \setminus S$ to denote the subgraph induced by $V(G) \setminus S$. The *open-neighborhood* of a vertex v in G, denoted $N(v)$, is the set of all vertices that are adjacent to v in G. The elements of $N(v)$ are said to be the *neighbors* of v, and $N[v] = N(v) \cup \{v\}$ is called the *closed neighborhood* of v. For a set of vertices $X \subseteq V(G)$, the open and closed neighborhoods of X are defined, respectively, as $N(X) = \bigcup_{u \in X} N(u) \setminus X$ and $N[X] = N(X) \cup X$. A vertex $v \in V(G)$ is said to be a *pendant* vertex of G if $|N(v)| = 1$. The girth of a graph is the size (number of vertices) of the smallest cycle in the graph. We use $\mathbb{G}_r$ to denote the class of all graphs with girth *at least* $r \in \mathbb{N}$.

A *dominating set* of graph G is a vertex-subset $S \subseteq V(G)$ such that for each $u \in V(G) \setminus S$ there exists $v \in S$ such that $\{u, v\} \in E(G)$. Given a graph G and $A, B \subseteq V(G)$, we say that A dominates B if every vertex in $B \setminus A$ is adjacent in G to some vertex in A. A *connected dominating set* of a graph $G = (V, E)$ is a set $S \subseteq V$ of vertices of G such that $G[S]$ is connected and S is a dominating set of G. To describe the running times of algorithms we sometimes use the $\mathcal{O}^*$ notation. The $\mathcal{O}^*$ notation suppresses polynomial factors in the expression.

A *parameterized problem* Π is a subset of $\Gamma^* \times \mathbb{N}$, where Γ is a finite alphabet. An instance of a parameterized problem is a tuple (x, k), where k is called the parameter. A central notion in parameterized complexity is *fixed-parameter tractability (FPT)* which means, for a given instance (x, k), decidability in time $\mathcal{O}(f(k) \cdot p(|x|))$, where f is an arbitrary function of k and p is a polynomial. The notion of *kernelization* is formally defined as follows.

▶ **Definition 1.** [**Kernelization, Kernel**] [16, 27] A kernelization algorithm for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, outputs, in time polynomial in $|x|+k$, a pair $(x', k') \in \Sigma^* \times \mathbb{N}$ such that (1) $(x, k) \in \Pi$ if and only if $(x', k') \in \Pi$ and (2) $|x'|, k' \leq g(k)$, where g is some computable function. The output instance x' is called the *kernel*, and the function g is called the size of the kernel. If $g(k) = k^{O(1)}$ then we say that Π admits a polynomial kernel.

3 On Graphs of Girth 3 and 4 : W[2]-Hardness

In [29, Theorem 1], it is shown that the closely related DS problem is W[2]-hard on graphs of girth 4. Their construction, reproduced below for completeness, suffices to show that CDS is W[2]-hard on graphs of girth 4:

▶ **Theorem 2.** CDS *is* W[2]-hard *on graphs of girth* 3 *and on graphs of girth* 4.

Proof. Given an instance (G, k) of DS, we construct a bipartite graph H. We take two copies of $V(G)$ call it $V_1 = \{u_1 \mid u \in V(G)\}$ and $V_2 = \{u_2 \mid u \in V(G)\}$. If there is an edge $\{u, v\}$ in E, then we add the edges $\{u_1, v_2\}$ and $\{v_1, u_2\}$ to H. We also include edges of the form $\{u_1, u_2\}$ for each $u \in V(G)$. We create two new vertices $z_1 \in V_1$ and $z_2 \in V_2$, and add an edge from every vertex in V_1 to z_2. This completes the construction of H.

The girth of the reduced instance H is at least 4 because H is bipartite, and H has girth exactly 4 because the reduction takes an edge in the original instance G to a cycle of length 4 in H. If G has a dominating set S of size at most k, then $S_1 = \{s_1 \in V_1 \mid s \in S\}$ and the vertex z_2 together form a connected dominating set of H of size at most $k+1$. For the reverse direction, observe that z_2 is present in any minimal connected dominating set of H. If D' is a connected dominating set of H of size at most $k+1$, then let $D = \{u \mid u \in V(G), u_1 \text{ or } u_2 \in D'\}$. It can easily be shown that D forms a dominating set of G of size at most k. It follows that the CDS problem restricted to graphs of girth 4 is W[2]-hard.

To see that CDS is W[2]-hard on graphs of girth 3 as well, add a new vertex z_3 and the two edges $\{z_2, z_3\}, \{z_1, z_3\}$ to H to form a triangle so that H has girth 3. The reduced instance is $(H, k+1)$. Essentially the same argument as above shows that this reduction is sound. ◀

4 On Graphs of girth 5 or More: $k^{\mathcal{O}(k)}$ kernel or FPT

We now show that the CDS problem restricted to $\mathbb{G}_5$ is FPT with an algorithm that runs in time $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$. A folklore theorem of parameterized complexity [27] states that for any computable function f, if a parameterized problem has an FPT algorithm that runs in time $f(k)n^{O(1)}$ on inputs of size n and parameter k, then the problem has a kernel of size $f(k)$. It follows that CDS restricted to $\mathbb{G}_5$ has a kernel of size $k^{\mathcal{O}(k)}$. We show first that a slightly more general problem is FPT on $\mathbb{G}_5$. Following [29], we define the Connected RWB-Dominating Set (ColCDS) problem as:

Connected RWB-Dominating Set(ColCDS)

Input:	A graph $G = (V, E)$, and a positive integer k. The vertex set of G is partitioned into three sets R, W, B of red, white, and blue vertices, respectively. In addition, G has the following properties: (a) G has girth at least 5; (b) every white vertex is the neighbor of some red vertex; (c) blue vertices have no red neighbors; and (d) $\|R\| \leq k$.
Parameter:	k
Question:	Does G have a connected dominating set of size at most k that contains all the red vertices?

The semantics of the colors are similar to those in [29]: A red vertex is one which is definitely present in the connected dominating set D that our algorithm is trying to construct. A white vertex is one that is not yet in D but is known to be dominated by some vertex in D. All the remaining vertices are those yet to be dominated and are colored blue.

We note that it is claimed in [29, Corollary 3] that CDS restricted to $\mathbb{G}_5$ has a kernel on $\mathcal{O}(k^3)$ vertices, and hence is fixed-parameter tractable. But the argument that they present is incorrect; in fact, as we show later (Theorem 13), CDS restricted to $\mathbb{G}_5$ *cannot* have any polynomial-sized kernel unless the Polynomial Hierarchy collapses to the third level. The error in their argument is that they assume that the reduction rules they used for DS also work for CDS — but rules like deleting a white vertex and edges between white vertices *do not* apply to CDS. This is because such vertices and edges may be needed to provide connectivity to a dominating set. However, the fixed-parameter tractability result still holds, as we prove by a different argument in the following lemma.

▶ **Lemma 3.** *ColCDS is FPT.*

Observe that once we have Lemma 3, we can solve the CDS problem on $\mathbb{G}_5$ by simply coloring all vertices blue and then solving the ColCDS problem using Lemma 3. Hence we have

▶ **Theorem 4.** *CDS is FPT on graphs of girth at least* 5.

Let (G, k) be an instance of ColCDS. If a vertex v in G has more than k neighbors, and v is not in a dominating set S of G of size at most k, then there is a vertex $u \in S$ that dominates at least two vertices $x, y \in N(v)$. Then u, v, x, y form a cycle of length at most 4, a contradiction. So we have:

▶ **Lemma 5.** *Let (G, k) be an instance of ColCDS. If a vertex v in G has more than k neighbors, then v is present in every dominating set of G of size at most k.*

Proof of Lemma 3. Let (G, k) be an instance of ColCDS and S be the set of white and blue vertices in G that have at least $k + 1$ neighbors. By Lemma 5 we know that every vertex of S is part of every dominating set of G size at most k whether connected or otherwise. Thus if $|R \cup S| > k$ then G does not have any connected dominating set of size at most k that contains all the vertices of R and hence we return NO. So we assume that $|R \cup S| \leq k$.

We first obtain an equivalent instance of ColCDS by coloring all the vertices of S red and all its blue neighbors white. Now we bound the size of the set B. Observe that in the equivalent instance every blue or white vertex has at most k neighbors and no red vertex has any blue neighbor. Thus the remaining $k' = k - |R|$ white and blue vertices can only dominate at most $k'(k + 1)$ blue vertices and hence $|B| \leq k^2 + k$ if (G, k) is a YES instance of the problem. So if $|B| > k^2 + k$, then we return NO.

Let W' be the set of white vertices that are neighbors to blue vertices. From Lemma 5, $|W'| \leq |B|k \leq k^3 + k^2$. Observe that every *connected dominating set* D of G of size at most k containing all the red vertices contains a *minimal dominating set* D' of size at most k such that $D' \subseteq B \cup W' \cup R$. This is because all the neighbors of B are in W'. We use this property to check whether G has a connected dominating set D of size at most k that contains all the red vertices. We enumerate all the minimal dominating sets D' of G of size at most k such that $R \subseteq D' \subseteq B \cup W' \cup R$. Given such a set D', we only need to check whether we can make it connected by adding at most $k - |D'|$ vertices. To do so we use an algorithm for the Steiner Tree problem. In the Steiner Tree problem we are given a graph G and a subset T of the vertex set called the terminal set, and the objective is to find a smallest set of vertices $N \subseteq V(G) \setminus T$ such that $G[T \cup N]$ is connected. Nederlof [26] gave an algorithm for Steiner Tree that runs in time $2^t n^{\mathcal{O}(1)}$ where $t = |T|$. Given D' we use this algorithm and check whether we can make D' connected by adding at most $k - |D'|$ vertices. If there is at least one D' such that we can connect it by adding at most $k - |D'|$ vertices, then we return YES, else we return NO. Note that $\ell = |B \cup W' \cup R| \leq (k^2 + k) + (k^3 + k^2) + k = \mathcal{O}(k^3)$.

Thus the running time of our algorithm is bounded by $\mathcal{O}^*(\sum_{i=|R|}^{k} \binom{\ell}{i} \cdot 2^i) = \mathcal{O}^*(2^k k^{3k})$. This concludes the proof of theorem. ◀

5 On Graphs of girth 5 and 6: No Polynomial Kernels

In the last section we saw that CDS is FPT on graphs with girth at least 5, with an algorithm of running time $k^{\mathcal{O}(k)}n^{\mathcal{O}(1)}$. This immediately implies that the problem has a kernel of size $k^{\mathcal{O}(k)}$[27]. A natural question to ask is whether CDS has polynomial kernels on these graph classes. We now show that the Connected Dominating Set problem restricted to graphs of girth 5 or 6 does not have a polynomial kernel unless the Polynomial Hierarchy collapses to the third level.

5.1 Known Lower Bound Machinery

To prove our lower bound, we need a few notions and results from the recently developed theory of kernel lower bounds [4, 6, 13]. We use a notion of reductions, similar in spirit to those used in classical complexity to show NP-hardness results, to show this kernelization lower bound. We recall the required definitions and theorems:

▶ **Definition 6.** [**Derived Classical Problem**] [6] Let $\Pi \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem, and let $1 \notin \Sigma$ be a new symbol. We define the *derived classical problem* associated with Π to be $\left\{x1^k \middle| (x,k) \in \Pi\right\}$.

▶ **Definition 7.** [**Composition Algorithm, Compositional Problem**] [4] A *composition algorithm* for a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is an algorithm that takes as input a sequence $\langle (x_1,k),(x_2,k),\ldots,(x_t,k)\rangle$ where each $(x_i,k) \in \Sigma^* \times \mathbb{N}$, runs in time polynomial in $\sum_{i=1}^{t} |x_i| + k$, and outputs an instance $(y,k') \in \Sigma^* \times \mathbb{N}$ where $(y,k') \in L \iff (x_i,k) \in L$ for some $1 \leq i \leq t$, and k' is polynomial in k. We say that a parameterized problem is *compositional* if it has a composition algorithm.

▶ **Theorem 8.** [4, Lemmas 1 and 2] *Let L be a compositional parameterized problem whose derived classical problem is NP-complete. If L has a polynomial kernel, then the Polynomial Hierarchy collapses to the third level.*

▶ **Definition 9.** [6] Let P and Q be parameterized problems. We say that P is polynomial parameter reducible to Q, written $P \leq_{ppt} Q$, if there exists a polynomial time computable function $f : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$, and a polynomial $p : \mathbb{N} \to \mathbb{N}$, and for all $x \in \Sigma^*$ and $k \in \mathbb{N}$, if $f((x,k)) = (x',k')$, then $(x,k) \in P$ if and only if $(x',k') \in Q$, and $k' \leq p(k)$. We call f a polynomial parameter transformation (or a PPT) from P to Q.

▶ **Theorem 10.** [6, Theorem 3] *Let P and Q be parameterized problems whose derived classical problems are P^c, Q^c, respectively. Let P^c be NP-complete, and $Q^c \in$ NP. Suppose there exists a PPT from P to Q. Then, if Q has a polynomial kernel, then P also has a polynomial kernel.*

5.2 Kernel lower bounds

We begin our reductions by defining the Fair Connected Colors problem, which is a variant of the Connected Colors problem recently introduced by Cygan et al. [9]:

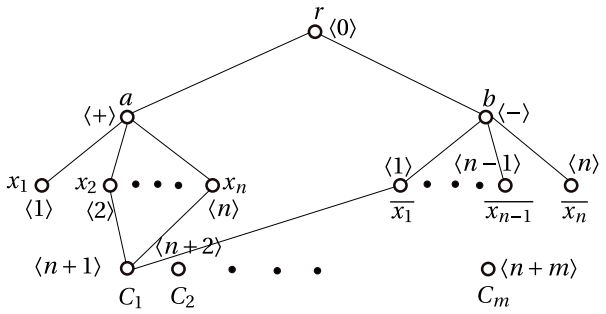


Figure 1 Reduction from CNF SAT to Fair Connected Colors. The color of each vertex is indicated within angled brackets.

Fair Connected Colors

Input:	A graph G, where the vertices $V(G)$ are *properly* colored with k colors in such a way that all neighbors of each vertex have distinct colors.
Parameter:	k
Question:	Does G contain a tree T on k vertices as a subgraph, where each vertex of T has a distinct color?

This problem differs from Connected Colors in that for Connected Colors, the given graph is *arbitrarily* colored with k colors. For Fair Connected Colors we restrict the coloring to be proper and fair (all neighbors of a vertex get different colors) as we need this restriction for the reduction we give in Theorem 13.

▶ **Lemma 11.** *The* Fair Connected Colors *problem is NP-complete.*

Proof. A tree on k vertices with all its vertices colored with distinct colors is a polynomial-time verifiable witness to a YES-instance of the problem, and so Fair Connected Colors is in NP. To show hardness, we reduce from the NP-complete CNF SAT problem [23]. Let ϕ be a Boolean formula in CNF on the variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$. We assume without loss of generality that there is no clause that contains both a variable and its negation. We construct a graph G on $m + 2n + 3$ vertices colored using $m + n + 3$ colors as follows: We define the vertex set to be $V(G) := \{r, a, b, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}, C_1, \ldots, C_m\}$. We add the edges $\{r, a\}, \{r, b\}$ and $\{a, x_1\}, \{a, x_2\}, \ldots, \{a, x_n\}, \{b, \overline{x_1}\}, \{b, \overline{x_2}\}, \ldots, \{b, \overline{x_n}\}$; and for each vertex C_i, we add an edge from C_i to vertex $y \in \{x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$ if and only if the literal y appears in clause C_i in the formula ϕ. This completes the construction of the graph G. We assign the colors $0, +, -$ to vertices r, a, b, respectively. For $1 \leq i \leq n$, we assign color i to vertices x_i and $\overline{x_i}$, and for $1 \leq j \leq m$, we assign color $n + j$ to vertex C_j. This completes the construction; see Figure 1.

Note that the vertices of G are properly colored with $n + m + 3$ colors in such a way that no vertex v is adjacent to two other vertices u, w where u and w are of the same color. The instance of Fair Connected Colors is $(G, n + m + 3)$. It remains to show that ϕ is satisfiable if and only if G contains an $m + n + 3$-vertex tree as a subgraph whose vertices are all colored distinctly.

Suppose ϕ is satisfiable, and let S be the set of literals (negative as well as positive) that are set to true by a satisfying assignment A of ϕ. Notice that A sets at least one literal in each clause of ϕ to true. Also, for each variable x_i, A sets exactly one of $x_i, \overline{x_i}$ to true. Thus each vertex $C_i; 1 \leq i \leq m$ is adjacent to at least one of vertex in S, and S contains exactly one vertex with each of the colors $\{1, 2, \ldots, n\}$. It follows that the subgraph H of G induced on the vertex set $\{r, a, b, C_1, C_2, \ldots, C_m\} \cup S$ is connected and has one vertex from each of

the $n+m+3$ colors $\{0,+,-,1,2,\ldots,n+m\}$. Therefore G contains an $m+n+3$-vertex tree as a subgraph whose vertices are all colored distinctly: indeed, any spanning tree of H serves as a witness.

Now suppose G contains an $m+n+3$-vertex tree T as a subgraph whose vertices are all colored distinctly. Then the vertex set $V(T)$ of T must consist of $\{r,a,b,C_1,\ldots,C_m\}$, and exactly n vertices from the set $X=\cup_{i=1}^{n}\{x_i,\overline{x_i}\}$ where exactly one vertex is chosen from $\{x_i,\overline{x_i}\}; 1\leq i\leq n$. The unique path from any vertex $C_i; 1\leq i\leq n$ to r in T must use a vertex in $S=X\cap V(T)$. Consider the assignment A of the formula ϕ which sets to true exactly those literals that appear in S. Since $|S\cap\{x_i,\overline{x_i}\}|=1$ for $1\leq i\leq n$, A is a valid assignment. Since each vertex C_i is adjacent to at least one vertex in S, the assignment satisfies every clause in ϕ, and so ϕ is satisfiable. ◀

The Fair Connected Colors problem is easily seen to be compositional: taking the disjoint union of input graphs suffices for the composition. That is, given k colored graph $G_1,\ldots,G_t$, return $\cup_{i=1}^{t}G_i$ and k. Hence from the Lemma 11 and Theorem 8 we have:

▶ **Lemma 12.** *The* Fair Connected Colors *problem does not have a polynomial kernel unless the Polynomial Hierarchy collapses to the third level.*

We now prove our main result by giving a polynomial parameter transformation (PPT) from Fair Connected Colors to CDS on graphs with graph 5 or 6.

▶ **Theorem 13.** *The CDS problem restricted to graphs of girth* 5 *or* 6 *does not admit a polynomial kernel unless the Polynomial Hierarchy collapses to the third level.*

Proof. Note that by Theorem 10 and Lemma 12 it is sufficient to show that there is a polynomial parameter transformation (PPT) from Fair Connected Colors to each of these problems. We first describe a PPT from Fair Connected Colors to CDS in graphs of girth six. Given an instance (G,k) of Fair Connected Colors, we construct an instance (H,k') of CDS where H has girth six and k' is bounded by a polynomial in k.

We start with a copy of G. For each color class (set of vertices of the same color) $\mathcal{C}_i$ of G, we add a new vertex v_i adjacent to all vertices of $\mathcal{C}_i$, and a new vertex g_i adjacent to v_i. The vertex g_i is essentially a guard vertex that will force v_i to be selected in our solution. We add a new vertex uv for each edge $\{u,v\}$ of G, and replace the edge $\{u,v\}$ by two new edges $\{u,uv\},\{uv,v\}$. That is, we split each edge of G once. For every two color classes $\mathcal{C}_i,\mathcal{C}_j; i<j$ of G,

1. We add two new vertices v_{ij} and g_{ij} and the edge $\{v_{ij},g_{ij}\}$.
2. For each edge $\{u,v\}$ in G where $u\in\mathcal{C}_i, v\in\mathcal{C}_j$, we add the edge $\{uv,v_{ij}\}$ where uv is the new vertex that splits $\{u,v\}$.
3. For each vertex $u\in\mathcal{C}_i$ that has no neighbor in $\mathcal{C}_j$, we add a new vertex u_{ij} and the edges $\{u,u_{ij}\},\{u_{ij},v_{ij}\}$ where v_{ij} is the vertex added in step 1.
4. Symmetrically, for each vertex $u\in\mathcal{C}_j$ that has no neighbor in $\mathcal{C}_i$, we add a new vertex u_{ji} and the edges $\{u,u_{ji}\},\{u_{ji},v_{ij}\}$.

This completes the construction of H; see Figure 2. For later reference, let S be the set of vertices of the form uv introduced in H to split the edges of G, $C=\mathcal{C}_1\cup\cdots\cup\mathcal{C}_k$, $X=\{g_i; 1\leq i\leq k\}$, $Y=\{v_{ij}\in V(H)\}$, $Z=\{v_1,v_2,\ldots,v_k\}$, $W=\{g_{ij}; 1\leq i<j\leq_k\}$, and let U be the set of all new vertices added in steps (3) and (4) above.

Observe that H is bipartite, with one part being $A=C\cup X\cup Y$. Hence every cycle in H is of even length, and the smallest cycle has length at least 4. Also, H contains a 4-cycle if and only if there are two vertices in A which have two common neighbors in $V(H)\setminus A$.

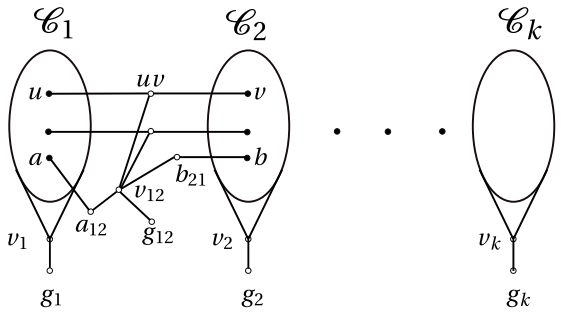


Figure 2 Reduction from Fair Connected Colors to Connected Dominating Set.

But no two vertices in A can have two common neighbors: the vertices in X are all of degree exactly one, and so they are not part of any cycle, and in each of the remaining ways of forming a pair a, b of vertices from A, it is easy to verify that a and b have at most one common neighbor. It follows that H does not contain a 4-cycle, and so the smallest cycle in H has length at least 6. To see that the girth of H is indeed 6, note that we can assume without loss of generality that $\mathcal{C}_1$ contains at least two vertices, say a, b. Observe that there is a path of length two from a to v_{12}, and a path of length two from v_{12} to b. These paths meet only at v_{12}, and together with the two edges $\{b, v_1\}, \{v_1, a\}$ they form a cycle of length 6. Thus let $(H, k^2 + k)$ be the reduced instance. Now we argue that the reduction is indeed sound.

Forward direction. Suppose G contains a tree T on k vertices, where each vertex of T has a distinct color. Let $V(T) = \{t_1, t_2, \ldots, t_k\}$, where $t_i \in \mathcal{C}_i$ for all i. Let T' be the "corresponding" tree in H: the vertex set of T' consists of $V(T)$ and all the new vertices in H that split the edges of T, and the edge set consists of all the new edges formed by splitting the edges of T. Thus T' is a tree on $2k - 1$ vertices. We now add more vertices and edges to T' to obtain a tree on $k^2 + k$ vertices that dominates all of H.

- For $1 \leq i \leq k$, we add the vertex v_i and the edge $\{v_i, t_i\}$ to T'. This adds k vertices.
- For $1 \leq i < j \leq k$, if the vertex $t_i t_j$ is present in T', then we add the vertex v_{ij} and the edge $\{t_i t_j, v_{ij}\}$ to T'. This adds $k - 1$ vertices to T'. Otherwise, let $a = t_i$. We add the vertices a_{ij}, v_{ij} and the edges $\{a, a_{ij}\}, \{a_{ij}, v_{ij}\}$ to T'. This adds two vertices for each "non-edge" in T, for a total of $2(\binom{k}{2} - (k-1))$ new vertices added to T'.

This completes the construction of T'. Note that T' is a tree on $4k-2+2(\binom{k}{2}-(k-1)) = k^2 + k$ vertices. In H, (1) the set $\{v_i \mid 1 \leq i \leq k\} \subseteq V(T')$ dominates all the vertices copied over from G, and the new vertices $\{g_1, \ldots, g_k\}$, and (2) the set $\{v_{ij} \mid 1 \leq i < j \leq k\} \subseteq V(T')$ dominates all the other newly added vertices. Thus T' is a connected dominating set of H on $k^2 + k$ vertices.

Reverse direction. Let D be a minimal connected dominating set of H with $1 < |D| \leq k^2 + k$. Observe first that vertices in $X \cup W$ are all pendant vertices, and all of their neighbors have degree at least 2. So $N(X \cup W) = (Y \cup Z) \subseteq D$, and since D is minimal, $D \cap (X \cup W) = \emptyset$. Now since $G[D]$ is connected and $|D| \geq 2$, at least one neighbor of each vertex in D must also be in D. Observe that for any two vertices $u, v \in Y \cup Z$, $N[u] \cap N[v] = \emptyset$, and so each vertex in D can be the neighbor of at most one vertex in $Y \cup Z \subseteq D$. Thus for each vertex $v \in Y \cup Z$, D contains at least one distinct vertex $u \in (N(v) \setminus (Y \cup Z))$, and so $|D| \geq 2|Y \cup Z| = 2(\binom{k}{2} + k) = k^2 + k$. But $|D| \leq k^2 + k$ by assumption, and so $|D| = k^2 + k$. Thus exactly one neighbor of each vertex in $Y \cup Z$ is in D. In particular, D contains exactly one vertex from each set $\mathcal{C}_i; 1 \leq i \leq k$. Further, $D = (Y \cup Z) \cup N(Y \cup Z)$.

Let T_1 be a spanning tree of $H[D]$. From the above arguments we see that all vertices in $Y \cup Z$ are leaves in T_1, and so $T_2 = T_1 \setminus (Y \cup Z)$ is also a tree. Observe that all the vertices in $V(T_2) \cap U$ are leaves in T_2, and so $T_3 = T_2 \setminus U$ is also a tree. Observe that T_3 consists of (1) exactly one vertex from each set $\mathcal{C}_i; 1 \leq i \leq k$, and (2) some vertices from the set S. Let T_4 be the tree obtained from T_3 by removing all those vertices in S that are leaves in T_3. Note that each vertex in $R = S \cap V(T_4)$ has degree exactly two in T_4, and no two vertices in R are adjacent in T_4. So the graph T obtained from T_4 by replacing each vertex $u \in R$ with an edge between the two neighbors of u is also a tree. From the construction, T is (isomorphic to) a subgraph of G. But T is a tree on k vertices where each vertex has a distinct color, and so (G, k) is a YES instance of Fair Connected Colors.

A small modification to the above reduction suffices to show that the Connected Dominating Set problem has no polynomial kernel in graphs of girth 5 as well, unless PH collapses: Add three new vertices a, b, c and the four new edges required to complete the 5-cycle v_1, a, b, c, g_1 so that H has girth 5. The reduced instance is $(H, k^2 + k + 2)$. In the argument to show that this reduction is sound, both the directions go through exactly as before once we observe that exactly one of the sets $\{v_1, a, g_1\}, \{v_1, a, b\}, \{v_1, g_1, c\}$ is contained in any minimal connected dominating set of H. ◀

6 On Graphs of girth 7 or More: A Cubic Kernel

We now show that CDS has a cubic kernel on graphs of girth at least 7. As before, our reduction rules color the vertices of G red, white, and blue. Red vertices are those that must necessarily be in any connected dominating set of G of size at most k. White vertices are those non-red vertices that are dominated by the red vertices, and blue vertices are the rest. Initially we color every vertex blue. We have the following four reduction rules.

(R1) Let S be the set of blue vertices in G that have at least $k+1$ blue neighbors. Color all the vertices of S red and all the blue neighbors in $N(S)$ white.

(R2) If $|R| > k$ or $|B| > k^2 + k$, then say NO and stop.

(R3) If G contains an isolated blue vertex, then say NO and stop.

(R4) If G contains a pendant blue or white vertex u adjacent to a vertex v, then remove u from G. If v is not red, then color v red and color all the remaining blue neighbors of v white.

Note that the class $\mathbb{G}_7$ is a subclass of $\mathbb{G}_5$. Hence the correctness of reduction rule (R1) is justified by Lemma 5. The bound obtained on $|B|$ in the proof of Lemma 3 justifies reduction rule (R2). Rule (R3) is justified as we need to include the isolated blue vertex in the dominating set (to dominate that vertex), but as it is isolated the dominating set will not induce a connected graph. Rule (R4) is justified as without loss of generality the vertex v can be in the minimal dominating set we are constructing (as u or v must be in any minimal dominating set to dominate u, and u is a pendant vertex).

From Rule (R2) we have that $|R| \leq k$ and $|B| \leq k^2 + k$. Now using the two additional rules and the fact that G has no cycles of length 5 or 6, we bound $|W|$.

▶ **Lemma 14.** *Let G be reduced with respect to the reduction rules (R1) to (R4) and let (G, k) be a* YES *instance of the* Connected Dominating Set *problem. Then $|W| \leq k^3 + \frac{5}{2}k^2 - \frac{3}{2}k$.*

Proof. We divide W into three parts, $W = W_B \cup W_R \cup W_W$, where

- W_B is the set of all white vertices that have *at least one* blue neighbor,
- W_R is the set of all white vertices in $W \setminus W_B$ that have *only* red neighbors, and
- W_W is the set of all white vertices $W \setminus W_B$ that have *at least one* white neighbor.

We now bound each of these sets.

By rule (R1) we know that any blue vertex v has degree at most k and hence can have at most k white neighbors. Thus $|W_B| \leq k|B| \leq k(k^2 + k)$.

Since G is reduced with respect to rule (R4) each vertex in W_R has at least two red neighbors. From this and the fact that no two vertices have more than one common neighbor, it follows that $|W_R| \leq \binom{|R|}{2} \leq \binom{k}{2}$. Note that we cannot just remove the vertices in W_R from G, since they could be useful in providing connectivity in some smallest connected dominating set.

Let E_W be the set of all edges $e \in E$ where both end vertices of e are white. Each white vertex is adjacent to some red vertex. For any pair of red vertices x, y, there is at most one edge $(u, v) \in E_W$ such that u is adjacent to x and v is adjacent to y. For, if there is another edge $(u', v') \in E_W$ where u' is adjacent to x and v' is adjacent to y, then the vertices x, y, u, v, u', v' form a cycle of length at most 6, a contradiction. It follows that $|E_W| \leq \binom{|R|}{2} \leq \binom{k}{2}$, and so $|W_W| \leq 2|E_W| \leq k^2 - k$.

Putting all the bounds together, if G has a connected dominating set of size at most k, then the number of white vertices in G is at most $k^3 + \frac{5}{2}k^2 - \frac{3}{2}k$. ◀

To obtain an (uncolored) instance of CDS, we now attach a new pendant vertex to each red vertex, and remove all colors to obtain an instance (G', k). This last step essentially "forces" all red vertices to be picked in any dominating set of G' of size at most k; it is easy to verify that this step is sound. From Lemma 14 and the bounds $|B| \leq k^2 + k$ and $|R| \leq k$, we get

▶ **Theorem 15.** *The* Connected Dominating Set *problem has a kernel on at most* $k^3 + \frac{7}{2}k^2 + \frac{3k}{2} = \mathcal{O}(k^3)$ *vertices on the class of graphs of girth at least* 7.

7 Conclusion

In this paper we studied the effect of excluding short cycles on CDS from the kernelization perspective. We obtained a very diverse kernelization landscape. The problem became progressively easier as the size of the girth increased with no kernels to polynomial kernels. It would be interesting to study other problems and excluding some other subgraphs. An interesting problem in this direction is whether CDS is FPT on claw-free graphs.

References

1. J. Alber, H. L. Bodlaender, H. Fernau, T. Kloks, and R. Niedermeier. Fixed parameter algorithms for dominating set and related problems on planar graphs. *Algorithmica*, 33(4):461–493, 2002.
2. J. Alber, M. R. Fellows, and R. Niedermeier. Polynomial-time data reduction for Dominating Set. *Journal of the ACM*, 51(3):363–384, 2004.
3. N. Alon and S. Gutner. Kernels for the Dominating Set Problem on Graphs with an Excluded Minor. Technical Report TR08-066, ECCC, 2008.
4. H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *Journal of Computer and System Sciences*, 75(8):423–434, 2009.
5. H. L. Bodlaender, F. V. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, and D. M. Thilikos. (Meta) Kernelization. In *Proceedings of FOCS 2009*, pages 629–638. IEEE, 2009.
6. H. L. Bodlaender, S. Thomassé, and A. Yeo. Kernel Bounds for Disjoint Cycles and Disjoint Paths. In *Proceedings of ESA 2009*, volume 5757 of *LNCS*, pages 635–646, 2009.

7 Y. Cao, J. Chen, and Y. Liu. On Feedback Vertex Set: New Measure and New Structures. In *Proceedings of SWAT 2010*, volume 6139 of *LNCS*, pages 93–104, 2010.

8 J. Chen, H. Fernau, I. A. Kanj, and G. Xia. Parametric Duality and Kernelization: Lower Bounds and Upper Bounds on Kernel Size. *SIAM Journal on Computing*, 37(4):1077–1106, 2007.

9 M. Cygan, M. Pilipczuk, M. Pilipczuk, and J. Wojtaszczyk. Kernelization hardness of connectivity problems in d-degenerate graphs. Accepted at WG 2010.

10 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Fixed-parameter algorithms for (k, r)-center in planar graphs and map graphs. *ACM Transactions on Algorithms*, 1(1):33–47, 2005.

11 E. D. Demaine, F. V. Fomin, M. Hajiaghayi, and D. M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *Journal of the ACM*, 52(6):866–893, 2005.

12 E. D. Demaine and M. Hajiaghayi. The Bidimensionality Theory and Its Algorithmic Applications. *The Computer Journal*, 51(3):332–337, 2007.

13 M. Dom, D. Lokshtanov, and S. Saurabh. Incompressibility through Colors and IDs. In *Proceedings of ICALP 2009*, volume 5555 of *LNCS*, pages 378–389. Springer, 2009.

14 F. Dorn, F. V. Fomin, D. Lokshtanov, V. Raman, and S. Saurabh. Beyond Bidimensionality: Parameterized Subexponential Algorithms on Directed Graphs. In *Proceedings of STACS 2010*, volume 5 of *LIPIcs*, pages 251–262, 2010.

15 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.

16 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

17 F. V. Fomin, D. Lokshtanov, S. Saurabh, and D. M. Thilikos. Bidimensionality and Kernels. In *Proceedings of SODA 2010*, pages 503–510, 2010.

18 F. V. Fomin and D. M. Thilikos. Dominating Sets in Planar Graphs: Branch-Width and Exponential Speed-Up. *SIAM Journal on Computing*, 36(2):281–309, 2006.

19 P. A. Golovach and Y. Villanger. Parameterized Complexity for Domination Problems on Degenerate Graphs. In *Proceedings of WG 2008*, volume 5344 of *LNCS*, 2008.

20 Q. Gu and N. Imani. Connectivity Is Not a Limit for Kernelization: Planar Connected Dominating Set. In *Proceedings of LATIN 2010*, volume 6034 of *LNCS*, pages 26–37, 2010.

21 S. Gutner. Polynomial Kernels and Faster Algorithms for the Dominating Set Problem on Graphs with an Excluded Minor. In *Proceedings of IWPEC 2009*, pages 246–257, 2009.

22 T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Fundamentals of Domination in Graphs*. CRC Press, 1998.

23 R. M. Karp. Reducibility among combinatorial problems. In *Complexity of Computer Communications*, pages 85–103, 1972.

24 D. Lokshtanov, M. Mnich, and S. Saurabh. Linear Kernel for Planar Connected Dominating Set. In *Proceedings of TAMC 2009*, volume 5532 of *LNCS*, pages 281–290, 2009.

25 N. Misra, G. Philip, V. Raman, S. Saurabh, and S. Sikdar. FPT Algorithms for Connected Feedback Vertex Set. In *Proceedings of WALCOM 2010*, volume 5942 of *LNCS*, pages 269–280, 2010.

26 J. Nederlof. Fast Polynomial-Space Algorithms Using Möbius Inversion: Improving on Steiner Tree and Related Problems. In *Proceedings of ICALP 2009*, 2009.

27 R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.

28 G. Philip, V. Raman, and S. Sikdar. Solving Dominating Set in Larger Classes of Graphs: FPT Algorithms and Polynomial Kernels. In *Proceedings of ESA 2009*, volume 5757 of *LNCS*, pages 694–705, 2009.

29 V. Raman and S. Saurabh. Short Cycles Make W-hard Problems Hard: FPT Algorithms for W-hard Problems in Graphs with no Short Cycles. *Algorithmica*, 52(2):203–225, 2008.

One-Counter Stochastic Games

Tomáš Brázdil[1], Václav Brožek[2], and Kousha Etessami[2]

1 **Faculty of Informatics, Masaryk University**
Botanická 68a, 602 00, Brno, Czech Republic
xbrazdil@fi.muni.cz

2 **School of Informatics, University of Edinburgh**
10 Crichton Street, EH8 9AB, Edinburgh, United Kingdom
{kousha,vbrozek}@inf.ed.ac.uk

Abstract

We study the computational complexity of basic decision problems for *one-counter simple stochastic games* (OC-SSGs), under various objectives. OC-SSGs are 2-player turn-based stochastic games played on the transition graph of classic one-counter automata. We study primarily the *termination* objective, where the goal of one player is to maximize the probability of reaching counter value 0, while the other player wishes to avoid this. Partly motivated by the goal of understanding termination objectives, we also study certain "limit" and "long run average" reward objectives that are closely related to some well-studied objectives for stochastic games with rewards. Examples of problems we address include: does player 1 have a strategy to ensure that the counter eventually hits 0, i.e., *terminates*, almost surely, regardless of what player 2 does? Or that the lim inf (or lim sup) counter value equals ∞ with a desired probability? Or that the long run average reward is > 0 with desired probability? We show that the *qualitative termination problem* for OC-SSGs is in $\mathbf{NP} \cap \mathbf{coNP}$, and is in P-time for 1-player OC-SSGs, or equivalently for *one-counter Markov Decision Processes* (OC-MDPs). Moreover, we show that *quantitative* limit problems for OC-SSGs are in $\mathbf{NP} \cap \mathbf{coNP}$, and are in $\mathbf{P}$-time for 1-player OC-MDPs. Both qualitative limit problems and qualitative termination problems for OC-SSGs are already at least as hard as Condon's quantitative decision problem for finite-state SSGs.

1998 ACM Subject Classification G.3; F.1.1; F.3.1

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.108

1 Introduction

There is a rich literature on the computational complexity of analyzing finite-state Markov decision processes and stochastic games. In recent years, there has also been some research done on the complexity of basic analysis problems for classes of finitely-presented but infinite-state stochastic models and games whose transition graphs arise from decidable infinite-state automata-theoretic models, including: context-free processes, one-counter processes, and pushdown processes (see, e.g., [9]). It turns out that such stochastic automata-theoretic models are intimately related to classic stochastic processes studied extensively in applied probability theory, such as (multi-type-)branching processes and (quasi-)birth-death processes (QBDs) (see [9, 8, 3]).

In this paper we continue this line of work by studying **one-counter simple stochastic games (OC-SSGs)**, which are turn-based 2-player zero-sum stochastic games on transition graphs of classic one-counter automata. In more detail, an OC-SSG has a finite set of control states, which are partitioned into three types: a set of *random* states, from where the next transition is chosen according to a given probability distribution, and states belonging to one of two players: *Max* or *Min*, from where the respective player chooses the next transition.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 108–119

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Transitions can change the state and can also change the value of the (unbounded) counter by at most 1. If there are no control states belonging to *Max* (*Min*, respectively), then we call the resulting 1-player OC-SSG a *minimizing* (*maximizing*, respectively) *one-counter Markov decision process* (OC-MDP).

Fixing strategies for the two players yields a countable state Markov chain and thus a probability space of infinite runs (trajectories). We focus in this paper on *objectives* that can be described by a (measurable) set of runs, such that player Max wants to maximize, and player Min wants to minimize, the probability of the objective. The central objective studied in this paper is *termination*: starting at a given control state and a given counter value $j > 0$, player Max (Min) wishes to maximize (minimize) the probability of eventually hitting the counter value 0 (in any control state).

Different objectives give rise to different computational problems for OC-SSGs, aimed at computing the value of the game, or optimal strategies, with respect to that objective. From general known facts about stochastic games (e.g., Martin's Blackwell determinacy theorem [14]), it follows that the games we study are *determined*, meaning they have a *value*: we can associate with each such game a *value*, ν, such that for every $\varepsilon > 0$, player Max has a strategy that ensures the objective is satisfied with probability at least $\nu - \varepsilon$ regardless of what player Min does, and likewise player Min has a strategy to ensure that the objective is satisfied with probability at most $\nu + \varepsilon$. In the case of termination objectives, the value may be *irrational* even when the input data contains only rational probabilities, and this is so even in the purely stochastic setting without any players, i.e., with only *random* control states (see [8]).

We can classify analysis problems for OC-SSGs into two kinds: *quantitative* analyses: "can the objective be achieved with probability at least/at most p" for a given $p \in [0, 1]$; or *qualitative* analyses, which ask the same question but restricted to $p \in \{0, 1\}$. We are often also interested in what kinds of strategies (e.g., memoryless, etc.) achieve these.

In a recent paper, [3], we studied *one-player* OC-SSGs, i.e., OC-MDPs, and obtained some complexity results for them under qualitative termination objectives and some quantitative limit objectives. The problems we studied included the qualitative termination problem (is the maximum probability of termination $= 1$?) for *maximizing* OC-MDPs. We showed that this problem is decidable in **P**-time. However, we left open the complexity of the same problem for *minimizing* OC-MDPs (is the minimum probability of termination < 1?). One of the main results of this paper is the following, which in particular resolves this open question:

▶ **Theorem 1. (Qualitative termination)** *Given a OC-SSG, $\mathcal{G}$, with the objective of termination, and given an initial control state s and initial counter value $j > 0$, deciding whether the value of the game is equal to 1 is in* ***NP*** $\cap$ ***coNP****. Furthermore, the same problem is in* ***P****-time for 1-player OC-SSGs, i.e., for both maximizing and minimizing OC-MDPs.*

Improving on this **NP** $\cap$ **coNP** upper bound for the qualitative termination problem for OC-SSGs would require a breakthrough: we show that deciding whether the value of an OC-SSG termination game is equal to 1 is already at least as hard as Condon's [6] *quantitative* reachability problem for finite-state simple stochastic games (Corollary 16). We do not know a reduction in the other direction. We furthermore show that if the value is 1 for a OC-SSG termination game, then Max has a simple kind of optimal strategy (memoryless, counter-oblivious, and pure) that ensures termination with probability 1, regardless of Min's strategy. Similarly, if the value is less than 1, we show Min has a simple strategy (using finite memory, linearly bounded in the number of control states) that ensures the probability of termination is $< 1 - \delta$ for some positive $\delta > 0$, regardless of what Max does. We show that such strategies for both players are computable in non-deterministic polynomial time

for OC-SSGs, and in deterministic P-time for (both maximizing and minimizing) 1-player OC-MDPs. We also observe that the analogous problem of deciding whether the value of a OC-SSG termination game is 0 is in **P**, which follows easily by reduction to non-probabilistic games.

OC-SSGs can be viewed as stochastic game extensions of Quasi-Birth-Death Processes (QBDs) (see [8, 3]). QBDs are a heavily studied model in queuing theory and performance evaluation (the counter keeps track of the number of jobs in a queue). It is very natural to consider controlled and game extensions of such queuing theoretic models, thus allowing for adversarial modeling of queues with unknown (non-deterministic) environments or with other unknown aspects modeled non-deterministically. OC-SSGs with termination objectives also subsume "solvency games", a recently studied class of MDPs motivated by modeling of a risk-averse investment scenario [1].

Due to the presence of an unbounded counter, an OC-SSG, $\mathcal{G}$, formally describes a stochastic game with a countably-infinite state space: a "configuration" or "state" of the underlying stochastic game consists of a pair (s, j), where s is a control state of $\mathcal{G}$ and j is the current counter value. However, it is easy to see that we can equivalently view $\mathcal{G}$ as a finite-state **simple stochastic game (SSG)**, $\mathcal{H}$, with **rewards** as follows: $\mathcal{H}$ is played on the finite-state transition graph obtained from that of $\mathcal{G}$ by simply ignoring the counter values. Instead, every transition t of $\mathcal{H}$ is assigned a *reward*, $r(t) \in \{-1, 0, 1\}$, corresponding to the effect that the transition t would have on the counter in $\mathcal{G}$. Furthermore, when emulating an OC-SSG using rewards, we can easily place rewards on states rather than on transitions, by adding suitable auxiliary control states. Thus, w.l.o.g., we can assume that OC-SSGs are presented as equivalent finite-state SSGs with a reward, $r(s) \in \{-1, 0, 1\}$ labeling each state s. A *run* of $\mathcal{H}$, w, is an infinite sequence of states that is permitted by the transition structure, and we denote the i-th state along the run w by $w(i)$. The termination objective for $\mathcal{G}$, when the initial counter value is $j > 0$, can now be rephrased as the following equivalent objective for $\mathcal{H}$:

$$\mathit{Term}(j) := \{\, w \mid w \text{ is a run of } \mathcal{H} \text{ such that there is } m > 0 \text{ such that } \textstyle\sum_{i=0}^{m} r(w(i)) = -j \,\}\,.$$

An important step toward our proof of Theorem 1 and related results, is to establish links between this termination objective and the following limit objectives, which are of independent interest. For $z \in \{-\infty, \infty\}$, and a comparison operator $\Delta \in \{>, <, =\}$, consider the following objective:

$$\mathit{LimInf}(\Delta z) := \{\, w \mid w \text{ is a run of } \mathcal{H} \text{ such that } \liminf_{n\to\infty} \textstyle\sum_{i=0}^{n} r(w(i)) \,\Delta\, z \,\}\,.$$

We will show that if j is large enough (larger than the number of control states), then the game value with respect to objective $\mathit{Term}(j)$ and the game value with respect to $\mathit{LimInf}(=-\infty)$ are either both equal to 1, or are both less than 1 (Lemma 14). We could also consider the "sup" variant of these objectives, such as $\mathit{LimSup}(=-\infty)$, but these are redundant. For example, by negating the sign of rewards, $\mathit{LimSup}(=-\infty)$ is "equivalent" to $\mathit{LimInf}(=+\infty)$. Indeed, the only limit objectives we need to consider for SSGs are $\mathit{LimInf}(=-\infty)$ and $\mathit{LimInf}(=+\infty)$, because the others are either the same objectives considered from the other player's points of view, or are vacuous, such as $\mathit{LimInf}(>+\infty)$. For both limit objectives, $\mathit{LimInf}(=-\infty)$ and $\mathit{LimInf}(=+\infty)$, we shall see that the value of the respective SSGs is always rational (Proposition 9). We shall also show that the objective $\mathit{LimInf}(=+\infty)$ is essentially equivalent to the following "mean payoff" objective (Lemma 10):

$$\mathit{Mean}(>0) := \{\, w \mid \text{w is a run of } \mathcal{H} \text{ such that } \liminf_{n\to\infty} \textstyle\sum_{i=0}^{n-1} r(w(i))/n > 0 \,\}\,.$$

This "intuitively obvious equivalence" is not so easy to prove. (Note also that $LimInf(=-\infty)$ is certainly not equivalent to $Mean(\leq 0)$.) We establish the equivalence by a combination of new methods and by using recent results by Gimbert, Horn and Zielonka [12, 13]. Mean payoff objectives are of course very heavily studied for stochastic games and for MDPs (see [16]). However, there is a subtle but important difference here: mean payoff objectives are typically formulated via *expected payoffs*: the Max player wishes to maximize the *expected* mean payoff, while the Min player wishes to minimize this. Instead, in the above $Mean(>0)$ objective we wish to maximize (minimize) the *probability* that the mean payoff is > 0. These require new algorithms. Our main result about such limit objectives is the following:

▶ **Theorem 2.** *For both limit objectives, $O \in \{LimInf(=-\infty), LimInf(=+\infty)\}$, given a finite-state SSG, $\mathcal{G}$, with rewards, and given a rational probability threshold, p, $0 \leq p \leq 1$, deciding whether the value of $\mathcal{G}$ with objective O is $>p$ (or $\geq p$) is in $\mathbf{NP} \cap \mathbf{coNP}$. If $\mathcal{G}$ is a 1-player SSG (i.e., a maximizing or minimizing MDP), then the game value can be computed in P-time.*

Although our upper bounds for both these objectives look the same, their proofs are quite different. We show that both players have pure and memoryless optimal strategies in these games (Proposition 7), which can be computed in P-time for 1-player (Max or Min) MDPs. Furthermore, we show that even deciding whether the value of these games is either 1 or 0, given input for which one of these two is promised to be the case, is already at least as hard as Condon's [6] *quantitative* reachability problem for finite-state simple stochastic games (Proposition 13). Thus, even any non-trivial *approximation* of the value of SSGs with such limit objectives is not easier than Condon's problem.

We already considered in [3] the problem of maximizing the probability of $LimInf(=-\infty)$ in a OC-MDP. There we showed that the maximum probability can be computed in P-time. However, again, we did not resolve the complementary problem of minimizing the probability of $LimInf(=-\infty)$ in a OC-MDP. Thus we could not address two-player OC-SSGs with either of these objectives, and we left these as key open problems, which we resolve here. An important distinction between *maximizing* and *minimizing* the probability of objective $LimInf(=-\infty)$ is that maximizing this objective satisfies a *submixing* property defined by Gimbert [11], which he showed implies the existence of optimal memoryless strategies, whereas minimizing the objective is not submixing, and thus we require new methods to tackle it, which we develop in this paper.

Finally, we mention that one can also consider OC-SSGs with the objective of terminating in a *selected* subset of states, F. Such objectives were considered for OC-MDPs in [3]. Using our termination results in this paper, we can also show that given an OC-SSG it is decidable (in double exponential time) whether Max can achieve a termination probability 1 in a selected subset of states, F. The computational complexity of selective termination is higher than for non-selective termination: PSPACE-hardness holds already for OC-MDPs without Min ([3]). Due to space limitations, we omit results about selective termination from this conference paper, and will include them in the journal version of this paper.

Related work.

As mentioned earlier, we initiated the study of some classes of 1-player OC-SSGs (i.e., OC-MDPs) in a recent paper [3]. The reader will find extensive references to earlier related literature in [3]. No earlier work considered OC-SSGs explicitly, but as we have highlighted already there are close connections between OC-SSGs and finite-state stochastic games with certain interesting limiting average reward objectives. One-counter automata with a non-negative counter are equivalent to pushdown automata restricted to a 1-letter stack alphabet

(see [8]), and thus OC-SSGs with the termination objective form a subclass of pushdown stochastic games, or equivalently, Recursive simple stochastic games (RSSGs). These more general stochastic games were introduced and studied in [9], where it is shown that many interesting computational problems for the general RSSG and RMDP models are undecidable, including generalizations of qualitative termination problems for RMDPs. It was also established in [9] that for stochastic context-free games (1-exit RSSGs), which correspond to pushdown stochastic games with only one state, both qualitative and quantitative termination problems are decidable, and in fact qualitative termination problems are decidable in $\mathbf{NP} \cap \mathbf{coNP}$ ([10]). Solving termination objectives is a key ingredient for many more general analyses and model checking problems for stochastic games. OC-SSGs form another natural subclass of RSSGs, which is incompatible with stochastic context-free games. Specifically, for OC-SSGs with the termination objective, the number of stack symbols, rather than the number of control states, of a pushdown stochastic game is being restricted to 1. As we show in this paper, this restriction again yields decidability of the qualitative termination problem. However, the decidability of the quantitative termination problem for OC-SSGs remains an open problem (see below).

Open problems.

Our results complete part of the picture for decidability and complexity of several problems for OC-SSGs. However, our results also leave many open questions. The most important open question for OC-SSGs is whether the *quantitative* termination problem, even for OC-MDPs, is decidable. Specifically, we do not know whether the following is decidable: given a OC-MDP, and a rational probability $p \in (0, 1)$, decide whether the maximum probability of termination is $>p$ (or $\geq p$). Substantial new obstacles arise for deciding this. In particular, we know that an optimal strategy may in general need to use different actions at the same control state for arbitrarily large counter values (so strategies cannot ignore the value of the counter, even for arbitrarily large values), and this holds already for the extremely simple case of solvency games [1, Theorem 3.7].

Outline of paper.

We fix notation and key definitions in Section 2. In Section 3, we prove Theorem 2. Building on Section 3, we prove Theorem 1 in Section 4. Due to space constraints, many proofs are only sketched here. Please refer to the full version [2] for missing details.

2 Preliminaries

▶ **Definition 3.** A **simple stochastic game (SSG)** is given by a finite, or countably infinite directed graph, $(V, \hookrightarrow)$, where V is the set of *vertices* (also called *states*), and $\hookrightarrow$ is the edge (also called *transition*) relation, together with a partition $(V_\top, V_\bot, V_P)$ of V, as well as a *probability assignment*, *Prob*, which to each $v \in V_P$ assigns a rational probability distribution on its set of outgoing edges. States in V_P are called *random*, states in $V_\top$ belong to player Max, and states in $V_\bot$ belong to Min. We assume that for all $v \in V$ there is some $u \in V$ such that $v \hookrightarrow u$. Writing $v \overset{x}{\hookrightarrow} u$ denotes $Prob(v \hookrightarrow u) = x$. If $V_\bot = \emptyset$ we call $\mathcal{G}$ a *maximizing* **Markov decision process (MDP)**. If $V_\top = \emptyset$ we call it a *minimizing* MDP. If $V_\bot = V_\top = \emptyset$ then we call $\mathcal{G}$ a **Markov chain**. A SSG (a MDP, a Markov chain) can be equipped with a reward function, r, which assigns to each state, $v \in V$, a number $r(v) \in \{-1, 0, 1\}$. Similarly, rewards can be assigned to transitions.

For a *path*, $w = w(0)w(1)\cdots w(n-1)$, of states in a graph, we use $len(w) = n$ to denote the length of w. A *run* in a SSG, $\mathcal{G}$, is an infinite path in the underlying directed graph. The set of all runs in $\mathcal{G}$ is denoted by $Run_{\mathcal{G}}$, and the set of all runs starting with a finite path w is $Run_{\mathcal{G}}(w)$. These sets generate the standard Borel algebra on $Run_{\mathcal{G}}$.

A *strategy* for player Max is a function, σ, which to each *history* $w \in V^+$ ending in some $v \in V_\top$, assigns a probability distribution on the set of outgoing transitions of v. We say that a strategy σ is *memoryless* if $\sigma(w)$ depends only on the last state, v, and *pure* if $\sigma(w)$ assigns probability 1 to some transition, for each history w. When σ is pure, we write $\sigma(w) = v'$ instead of $\sigma(w)(v, v') = 1$. Strategies for player Min are defined similarly, just by substituting $V_\top$ with $V_\bot$.

For every starting state s, and a pair of strategies: σ for player Max, and π for Min in a SSG, $\mathcal{G}$, there is a unique probabilistic measure, $\mathbb{P}_s^{\sigma,\pi}$, on the Borel space of runs $Run_{\mathcal{G}}$, satisfying for all finite paths w starting in s: $\mathbb{P}_s^{\sigma,\pi}(Run_{\mathcal{G}}(w)) = \prod_{i=1}^{len(w)-1} x_i$ where x_i, $1 \leq i < len(w)$ are defined by requiring that (a) if $w(i{-}1) \in V_P$ then $w(i{-}1) \xrightarrow{x_i} w(i)$; and (b) if $w(i{-}1) \in V_\top$ then $\sigma(w(0)\cdots w(i{-}1))$ assigns x_i to the transition $w(i{-}1) \hookrightarrow w(i)$; and (c) if $w(i{-}1) \in V_\bot$ then $\pi(w(0)\cdots w(i{-}1))$ assigns x_i to the transition $w(i{-}1) \hookrightarrow w(i)$. Note that $\mathbb{P}_s^{\sigma,\pi}(Run_{\mathcal{G}}(s)) = 1$. If $\mathcal{G}$ is a maximizing MDP, a minimizing MDP, or a Markov chain, we denote this probability measure by $\mathbb{P}_s^{\sigma}$, $\mathbb{P}_s^{\pi}$, or $\mathbb{P}_s$, respectively. See, e.g., [16, p. 30], for the existence and uniqueness of the measure $\mathbb{P}_s^{\sigma}$ in the case of MDPs. Consider pairs of strategies to be one strategy to establish existence and uniqueness of $\mathbb{P}_s^{\sigma,\pi}$ for SSGs.

In this paper, an *objective* for a stochastic game is given by a measurable set of runs. An objective, O, is called a *tail* objective if for all runs w and all suffixes w' of w, we have $w' \in O \iff w \in O$. Assume we have fixed a SSG, an objective, O, and a starting state, s. We define the *value of* $\mathcal{G}$ *in* s as $Val^O(s) := \sup_\sigma \inf_\pi \mathbb{P}_s^{\sigma,\pi}(O)$. It follows from Martin's Blackwell determinacy theorem [14] that these games are *determined*, meaning $Val^O(s) = \inf_\pi \sup_\sigma \mathbb{P}_s^{\sigma,\pi}(O)$. A strategy σ for Max is *optimal in* s if $\mathbb{P}_s^{\sigma,\pi}(O) \geq Val^O(s)$ for every π. Similarly a strategy π for Min is *optimal in* s if $\mathbb{P}_s^{\sigma,\pi}(O) \leq Val^O(s)$ for every σ. A strategy is called *optimal* if it is optimal in every state. An important objective for us is *reachability*. Given a set $T \subseteq V$, we define the objective $Reach(T) := \{w \in Run_{\mathcal{G}} \mid \exists i \geq 0 : w(i) \in T\}$. The following fact is well known:

▶ **Fact 4.** *(See, e.g., [16, 6, 7].) For both maximizing and minimizing finite-state MDPs with reachability objectives, pure memoryless optimal strategies exist and can be computed, together with the optimal value, in polynomial time.*

3 Limit objectives

All MDPs and SSGs in this section have finitely many states. Rewards are assigned to states, not to transitions. The main goal of this section is to prove Theorem 2. We start by proving that both players have optimal pure and memoryless strategies for objectives $LimInf(=-\infty)$, $LimInf(=+\infty)$, and $Mean(>0)$. The following is a corollary of a result by Gimbert and Zielonka, which allows us to concentrate on MDPs instead of SSGs:

▶ **Fact 5.** *(See [13, Theorem 2].) Fix any objective, O, and suppose that in every maximizing and minimizing MDP with objective O, the unique player has a pure memoryless optimal strategy. Then in all SSGs with objective O, both players have optimal pure and memoryless strategies.*

Note that the probability of $LimInf(=-\infty)$ is minimized iff the probability of $LimInf(>-\infty)$ is maximized, similarly with $LimInf(=+\infty)$ vs. $LimInf(<+\infty)$, and $Mean(>0)$ vs. $Mean(\leq 0)$.

▶ **Fact 6.** *(See [12, Theorem 4.5].) Let O be a tail objective. Assume that for every maximizing MDP and for every state, s, with $Val^O(s) = 1$, there is an optimal pure memoryless strategy starting in s. Then for all s there is an optimal pure memoryless strategy starting in s, without restricting $Val^O(s)$.*

▶ **Proposition 7.** *For every SSG, with any of the objectives $LimInf(=-\infty)$, $LimInf(=+\infty)$, or $Mean(>0)$, both players Max and Min have optimal pure memoryless strategies.*

Proof. *(Sketch.)* Using Fact 5 we consider only maximizing MDPs, and prove the proposition for the objectives listed and their complements. Note that since all these objectives are tail, a play under an optimal strategy, starting from a state with value 1, cannot visit a state with value < 1. By Fact 6 we may thus safely assume that the value is 1 in all states. We discuss different groups of objectives:
$LimInf(=-\infty)$, $LimInf(<+\infty)$, $Mean(\leq 0)$, $Mean(>0)$: The first three (with $LimInf(=-\infty)$ also handled explicitly in [3]) are *tail* objectives and are also *submixing* (see [11]). Therefore, Theorem 1 of [11] immediately yields the desired result. $Mean(>0)$ can be equivalently rephrased via a submixing lim sup variant. See [2] for details.
$LimInf(=+\infty)$: is a tail objective, so there is always a pure optimal strategy, τ, by [12, Theorem 3.1]. Note that $LimInf(=+\infty)$ is *not submixing*, so Theorem 1 of [11] does not apply. In the following we proceed in two steps: we start with τ and convert it to a finite-memory strategy[1], σ. Finally, we reduce the use of memory to get a memoryless strategy.

First, we obtain a finite-memory optimal strategy, starting in some state, s. For a run $w \in Run_{\mathcal{G}}(s)$ and $i \geq 0$, we denote by $r[i](w)$ the accumulated reward $\sum_{j=0}^{i} r(w(j))$ up to step i. Observe that because τ is optimal there is some $m > 0$ and a (measurable) set of runs $A \subseteq Run_{\mathcal{G}}(s)$, such that $\mathbb{P}_s^\tau(A) \geq \frac{1}{2}$, and for all $w \in A$ we have that the accumulated reward along w never reaches $-m$ (i.e. $\inf_{i \geq 0} r[i](w) > -m$). Since for almost all runs of A we have $\lim_{i \to \infty} r[i](w) = \infty$, there is some $n > 0$ and a set $B \subseteq A$ such that $\mathbb{P}_s^\tau(B \mid A) \geq \frac{1}{2}$ (and hence, $\mathbb{P}_s^\tau(B) \geq \frac{1}{4}$), and for all $w \in B$ we have that the accumulated reward along w reaches $4m$ before the n-th step. Thus with probability at least $\frac{1}{4}$, a run $w \in Run_{\mathcal{G}}(s)$ satisfies $\inf_{i \geq 0} r[i](w) > -m$ and $\max_{0 \leq i \leq n} r[i](w) \geq 4m$.

We denote by $T_s(w)$ the *stopping time* over $Run_{\mathcal{G}}(s)$ which for every $w \in Run_{\mathcal{G}}(s)$ returns the least number $i \geq 0$ such that either $r[i](w) \notin (-m, 4m)$, or $i = n$. Observe that the expected accumulated reward at the stopping time T_s is at least $\frac{1}{4} \cdot 4m + \frac{3}{4}(-m) = \frac{m}{4} > 0$. Let us define a new strategy σ as follows. Starting in a state $s \in V$, the strategy σ chooses the same transitions as τ started in s, up to the stopping time T_s. Once the stopping time is reached, say in a state v, the strategy σ erases its memory and behaves like τ started anew in v. Subsequently, σ follows the behavior of τ up to the stopping time T_v. Once the stopping time T_v is reached, say in a state u, σ erases its memory and starts to behave as τ started anew in u, and so on. Observe that the strategy σ uses only finite memory because each stopping time T_s is bounded for every state s. Because τ is pure, so is σ.

Now we argue that σ is optimal. Intuitively, this is because, on average, the accumulated reward strictly increases between resets of the memory of σ. To formally argue that this implies that the accumulated reward increases indefinitely, we employ the theory of random walks on $\mathbb{Z}$ and sums of i.i.d. random variables (see, e.g., Chapter 8 of [5]). Essentially, we define a set of random walks, one for each state s, capturing the sequence of changes to the accumulated reward between each reset in s and the next reset (in any state). We can then

[1] A finite-memory strategy is specified by a finite state automaton, $\mathcal{A}$, over the alphabet V. Given $w \in V^+$, the value $\sigma(w)$ is determined by the state of $\mathcal{A}$ after reading w.

apply random walk results, e.g., from [5, Chapter 8], to conclude that these walks diverge to ∞ almost surely. Details are given in [2].

Taking the product of the finite-memory strategy σ and $\mathcal{G}$ yields a finite-state Markov chain. By analyzing its bottom strongly connected components we can eliminate the use of memory, and obtain a pure and memoryless optimal strategy. See [2] for details.

LimInf$(>-\infty)$: Like *LimInf*$(=+\infty)$, the objective *LimInf*$(>-\infty)$ is tail, but not submixing. Thus there is always a pure optimal strategy, τ, for *LimInf*$(>-\infty)$, by [12, Theorem 3.1], but Theorem 1 of [11] does not apply. We will prove Proposition 7 for *LimInf*$(>-\infty)$ using the results for *LimInf*$(=+\infty)$, and also a new objective, $All(\geq 0) := \{w \in Run_{\mathcal{G}} \mid \forall n \geq 0 : \sum_{j=0}^{n} r(w(j)) \geq 0\}$. Let W_∞ and W_+ denote the sets of states s such that $Val^{LimInf(=+\infty)}(s) = 1$, and $Val^{All(\geq 0)}(s) = 1$, respectively. The following is true for every state, s, with $Val^{LimInf(>-\infty)}(s) = 1$ (see [2] for details):

$$\exists \sigma : \mathbb{P}_s^\sigma(Reach(W_\infty \cup W_+)) = 1 \tag{1}$$

Moreover, we prove that whenever $Val^{All(\geq 0)}(s) = 1$ then Max has a pure and memoryless strategy σ_+ which is optimal in s for $All(\geq 0)$. Indeed, observe that player Max achieves $All(\geq 0)$ with probability 1 iff *all* runs satisfy it. So we may consider the MDP $\mathcal{G}$ as a 2-player non-stochastic game, where random nodes are now treated as player Min's. In this case, Theorem 12 of [4] guarantees the existence of the promised strategy σ_+. The proof is now finished by observing that, by Fact 4, there is a pure and memoryless strategy σ maximizing the probability of reaching $W_\infty \cup W_+$. The resulting pure and memoryless strategy, optimal for *LimInf*$(>-\infty)$, can be obtained by "stitching" σ together with the respective optimal strategies for *LimInf*$(=+\infty)$ and $All(\geq 0)$. ◀

▶ **Lemma 8** (see [2]). *Let $\mathcal{M}$ be a finite, strongly connected (irreducible) Markov chain, and O be a tail objective. Then there is $x \in \{0, 1\}$ such that $\mathbb{P}_s(O) = x$ for all states s.*

A corollary of the previous proposition and lemma is the following:

▶ **Proposition 9.** *Let $O \in \{LimInf(=-\infty), LimInf(=+\infty), Mean(>0)\}$. Then in every SSG, and for all states, s, $Val^O(s)$ is rational, with a polynomial length binary encoding.*

Proof. By Proposition 7, there are memoryless optimal strategies: σ for Max, and π for Min. Fixing them induces a Markov chain on the states of $\mathcal{G}$. By Lemma 8, in every fixed bottom strongly connected component (BSCC), C, of this Markov chain, all states $v \in C$ have the same value, x_C, which is either 0 or 1. Denote by W the union of all BSCCs, C, with $x_C = 1$. By optimality of σ and π, $Val^O(s) = \mathbb{P}_s^{\sigma,\pi}(Reach(W))$ for every $s \in V$. By, e.g., [7, Section 3], this probability is rational, with polynomial length bit encoding, since reaching W is a regular event, and every Markov chain is a special case of a MDP. ◀

Proof of Theorem 2.

▶ **Lemma 10.** *Let $\mathcal{G}$ be a MDP with rewards, and s a state of $\mathcal{G}$. Then for every memoryless strategy σ: $\mathbb{P}_s^\sigma(Mean(>0)) = \mathbb{P}_s^\sigma(LimInf(=+\infty))$. In particular, both objectives are equivalent with respect to both the value and optimal strategies.*

Proof. *(Sketch.)* The inequality $\leq$ is true for all strategies, since $Mean(>0) \subseteq LimInf(=+\infty)$. In the other direction, the property that σ is memoryless is needed, so that fixing σ yields a Markov chain on the states of $\mathcal{G}$. In this Markov chain, by Lemma 8, for every BSCC, C, there are $x_C \leq y_C \in \{0, 1\}$, such that $\mathbb{P}_s^\sigma(Mean(>0) \mid Reach(C)) = x_C$, and $\mathbb{P}_s^\sigma(LimInf(=+\infty) \mid Reach(C)) = y_C$. By random walk arguments, considering the rewards

accumulated between subsequent visits to a fixed state in C, we can prove that $y_C = 1 \implies x_C = 1$, see [2] for details. Proposition 7 finishes the proof. ◀

▶ **Lemma 11.** *For an objective $O = LimInf(=-\infty)$, $LimInf(>-\infty)$, $LimInf(=+\infty)$, or $LimInf(<+\infty)$, and a maximizing MDP, $\mathcal{G}$, denote by W the set of all $s \in V$ satisfying $Val^O(s) = 1$. Then $Val^O(s) = Val^{Reach(W)}(s)$ for every state s.*

Proof. Proposition 7 gives us a memoryless optimal strategy, σ. Fix it and obtain a Markov chain on states of $\mathcal{G}$. Denote by W' the union of all BSCCs in which at least one state has a positive value. By Lemma 8, all states from W' have, in fact, value 1. Since $W' \subseteq W$, and σ is optimal, we get $Val^O(s) = \mathbb{P}^\sigma_s(O) = \mathbb{P}^\sigma_s(Reach(W')) \leq \mathbb{P}^\sigma_s(Reach(W)) \leq Val^{Reach(W)}(s)$ for every state s. As O is a tail objective, we easily obtain $Val^O(s) \geq Val^{Reach(W)}(s)$. ◀

To prove Theorem 2, we start with the MDP case. By Proposition 7, pure memoryless strategies are sufficient for optimizing the probability of all the objectives considered in this theorem, so we can restrict ourselves to such strategies for this proof. Given an objective O, we will write W^O to denote the set of states s with $Val^O(s) = 1$. As $\mathcal{G}$ is a MDP, optimal strategies for *reaching* any state in W^O can be computed in polynomial time, by Fact 4. If O is any of the objectives mentioned in the statement of Lemma 11, then by that Lemma, in order to compute optimal strategies and values for objective O, it suffices to compute the set W^O and optimal strategies for the objective O in states in W^O. The resulting optimal strategy "stitches" these and the optimal strategy for reaching W^O.

▶ **Proposition 12.** *For every MDP, $\mathcal{G}$, and an objective $O = LimInf(=-\infty)$, $LimInf(=+\infty)$, or $Mean(>0)$, the problem whether $s \in W^O$ is decidable in $\boldsymbol{P}$-time. If $s \in W^O$, then a strategy optimal in s is computable in $\boldsymbol{P}$-time.*

Proof. *(Sketch.)* From Lemma 10 we know that $LimInf(=+\infty)$ is equivalent to $Mean(>0)$, and thus we only have to consider $O = LimInf(=-\infty)$ and $O = Mean(>0)$. For a uniform presentation, we assume that $\mathcal{G}$ is a maximizing MDP, and consider two cases: $O = Mean(>0)$, and $LimInf(>-\infty)$. The remaining cases were solved in [3] – Theorem 3.1 there solves the case $O = LimInf(=-\infty)$, and Section 3.3 solves $O = Mean(\leq 0)$.
$O = Mean(>0)$: We design an algorithm to decide whether $\max_\sigma \mathbb{P}^\sigma_s(Mean(>0)) = 1$, using the existing polynomial time algorithm, based on linear programming, for maximizing the *expected* mean payoff and computing optimal strategies for it (see, e.g., [16]). Note that it does not matter whether $\liminf$ or $\limsup$ is used in the definition of $Mean(>0)$ (see [2] for details). Under a memoryless strategy σ, almost all runs in $\mathcal{G}$ reach one of the bottom strongly connected components (BSCCs). Almost all runs initiated in some BSCC, C, visit all states of C infinitely often, and it follows from standard Markov chain theory (e.g., [15]) that almost all runs in C have the same mean payoff, which equals the expected mean payoff for the Markov chain induced by C.

The algorithm is given here as Procedure `MP`(s). Both step 2, as well as verifying the condition from step 4, can be done in P-time, because, as observed above, this is equivalent to verifying that the expected mean payoff in C is positive, which can be done in P-time (see [16, Theorem 9.3.8]). Step 5 can be done in P-time by Fact 4. To obtain a formally correct MDP, we introduce a new state z with a self-loop, and after the removal of any state v in step 7 of the for loop, we redirect all stochastic transitions leading to v to this new state z, and eliminate all other transitions into v. The reward of the new state z is set to 0. This will not affect the sign of subsequent optimal expected mean payoffs starting from s, unless s has been already removed. Thus, the algorithm can be implemented so that each iteration of the repeat-loop takes P-time, and so the algorithm terminates in P-time, since in each

Procedure `MP(`s`)`

Data: A state s.
Result: Decide $Val^{Mean(>0)}(s) \stackrel{?}{=} 1$. If yes, return a strategy σ with $\mathbb{P}^{\sigma}_{s}(Mean(>0)) = 1$.

1 **repeat**
2 Compute a strategy σ_{mp} maximizing the expected mean payoff.
3 **if** $\mathbb{E}^{\sigma_{mp}}_{s}(\textit{mean payoff}) \leq 0$ **then return** No
4 Fix σ_{mp} to get a Markov chain on $\mathcal{G}$. Find a BSCC, C, with mean payoff almost surely positive.
5 Compute a strategy σ_C maximizing the probability of $Reach(C)$.
6 **foreach** v *with* $\mathbb{P}^{\sigma_C}_{v}(Reach(C)) = 1$ **do**
7 Remove state v.
8 **if** $v \in C$ **then** $\sigma(v) \leftarrow \sigma_{mp}(v)$ **else** $\sigma(v) \leftarrow \sigma_C(v)$
9 **until** s *is cut off*
10 **return** (Yes, σ)

iteration at least one state must be removed. If the algorithm outputs (Yes, σ) then clearly $\mathbb{P}^{\sigma}_{s}(Mean(>0)) = 1$. On the other hand, by an easy induction on the number of iterations of the repeat-loop one can prove that if $Val^{Mean(>0)}(s) = 1$ then the following is an invariant of line 9: either s has been removed, or the maximal expected mean payoff starting in s is positive. In particular, the algorithm cannot output No. Thus we have completed the case when $O = Mean(>0)$.

$O = LimInf(>-\infty)$: Recall first the auxiliary objective $All(\geq 0) := \{w \in Run_{\mathcal{G}} \mid \forall n \geq 0 : \sum_{j=0}^{n} r(w(j)) \geq 0\}$ from the proof of Proposition 7, and also the sets $W_{\infty} = \{v \mid Val^{LimInf(=+\infty)}(v) = 1\}$, and $W_{+} = \{v \mid Val^{All(\geq 0)}(v) = 1\}$. Note that $W_{\infty} = W^{Mean(>0)}$, by Lemma 10. Finally, recall from the equation (1) in the proof of Proposition 7, that the probability of $LimInf(>-\infty)$ is maximized by almost surely reaching $W_{\infty} \cup W_{+}$ and then satisfying $All(\geq 0)$ or $LimInf(=+\infty)$. We note that the strategy σ_{+}, optimal for $All(\geq 0)$, from the proof of Proposition 7, can be computed in polynomial time by [4, Theorem 12]. The results on $Mean(>0)$ and Fact 4 conclude the proof. ◀

Now we finish the proof of Theorem 2. Proposition 12 and Fact 4 together establish the MDP case. Establishing the NP ∩ coNP upper bound for SSGs proceeds in a standard way: guess a strategy for one player, fix it to get a MDP, and verify in polynomial time (Proposition 12) that the other player cannot do better than the given value p. To decide whether, e.g., $Val^{O}(s) \geq p$, guess a strategy σ for Max, fix it to get an MDP, and verify that Min has no strategy π so that $\mathbb{P}^{\sigma,\pi}_{s}(O) < p$. Other cases are similar.

Finally, we show that the upper bound from Theorem 2 is hard to improve upon:

▶ **Proposition 13.** *Assume that a SSG, $\mathcal{G}$, a state s, and a reward function r are given, and let $O = LimInf(=-\infty)$, $LimInf(=+\infty)$, or $Mean(>0)$. Moreover, assume the property (promise) that either $Val^{O}(s) = 1$ or $Val^{O}(s) = 0$. Then deciding which is the case is at least as hard as Condon's [6]* quantitative *reachability problem w.r.t. polynomial time reductions.*

Proof. The problem studied by Condon [6] is: given a SSG, $\mathcal{H}$, an initial state s, and a target state t, decide whether $Val^{Reach(t)}(s) \geq 1/2$. Deciding whether $Val^{Reach(()t)}(s) > 1/2$ is P-time equivalent. Moreover, we may safely assume there is a state $t' \neq t$, such that whatever strategies are employed, we reach t or t', with probability 1. Consider the following reduction:

given a SSG, $\mathcal{H}$, with distinguished states s, t, and t' as above, produce a new SSG, $\mathcal{G}$, with rewards as follows: remove all outgoing transitions from t and t', add transitions $t \hookrightarrow s$ and $t' \hookrightarrow s$, and make both t and t' belong to Max. Let r be the reward function over states of $\mathcal{G}$, defined as follows: $r(t) := -1$, $r(t') := +1$ and $r(z) := 0$ for all other $z \notin \{t, t'\}$. It follows from basic random walk theory that in $\mathcal{G}$, $Val^{LimInf(=-\infty)}(s) = 1$ if $Val^{Reach(t)}(s) \geq 1/2$, and $Val^{LimInf(=-\infty)}(s) = 0$ otherwise. Likewise, $Val^{LimInf(=+\infty)}(s) = 1$ if $Val^{Reach(t')}(s) > 1/2$, and $Val^{LimInf(=+\infty)}(s) = 0$ otherwise, and identically for the objective $Mean(>0)$ which we already showed to be equivalent to $LimInf(=+\infty)$. ◀

4 Termination

Here we prove Theorem 1. We continue viewing OC-SSGs as finite-state SSGs with rewards, as discussed in the introduction. However, for notational convenience this time we consider rewards on *transitions* rather than on states. It is easy to observe that Theorem 2 remains valid even if we sum rewards on transitions instead of rewards on states in the definition of $LimInf(=-\infty)$. We fix a SSG, $\mathcal{G}$, with state set V, and a reward function r.

▶ **Lemma 14.** *For all states s and $j \geq |V|$: $Val^{Term(j)}(s) = 1$ iff $Val^{LimInf(=-\infty)}(s) = 1$.*

Proof. If $\mathcal{G}$ is a maximizing MDP, the proposition is true by results of [3, Section 4]. Consider now the general case, when $\mathcal{G}$ is a SSG. If $Val^{LimInf(=-\infty)}(s) = 1$ then clearly $Val^{Term(j)}(s) = 1$. Now assume that $Val^{Term(j)}(s) = 1$ and consider the memoryless strategy of player Min, optimal for $LimInf(=-\infty)$, which exists by Proposition 7. Fixing it, we get a maximizing MDP, in which the value of $Term(j)$ in s is, of course, still 1. We already know from the above discussion that the value of $LimInf(=-\infty)$ in s is thus also 1 in this MDP. Since the fixed strategy for Min was optimal, we get that $Val^{LimInf(=-\infty)}(s) = 1$ in $\mathcal{G}$. Thus, if $Val^{Term(j)}(s) = 1$ then $Val^{LimInf(=-\infty)}(s) = 1$. ◀

Proof of Theorem 1.

For cases where $j \geq |V|$, the theorem follows directly from Lemma 14 and Theorem 2. If $j < |V|$ then we have to perform a simple reachability analysis, similar to the one presented in [3]. The following SSG, $\mathcal{G}'$, keeps track of the accumulated rewards as long as they are between $-j$ and $|V| - j$: its set of states is $V' := \{(u, i) \mid u \in V, -j \leq i \leq |V| - j\}$.

States (u, i) with $i \in \{-j, |V| - j\}$ are absorbing, and for $i \notin \{-j, |V| - j\}$ we have $(u, i) \to (t, k)$ iff $u \to t$ and $k = i + r(u \to t)$. Every (u, i) belongs to the player who owned u. The probability of every transition $(u, i) \to (t, k)$, $u \in V_P$, is the same as that of $u \to t$. There is no reward function for $\mathcal{G}'$, we consider a reachability objective instead, given by the target set $R := \{(u, -j) \mid u \in V\} \cup \{(u, i) \mid -j \leq i \leq |V| - j, Val^{LimInf(=-\infty)}(u) = 1\}$. Finally, let us observe that, by Lemma 14, $Val^{Reach(R)}((s, 0)) = 1$ iff $Val^{Term(j)}(s) = 1$. Since the size of $\mathcal{G}'$ is polynomial in the size of $\mathcal{G}$, Theorem 1 is proved.

▶ **Proposition 15.** *For all $j > 0$, $s \in V$, there are pure strategies, σ for Max, and π for Min, such that*

1. *If $Val^{Term(j)}(s) = 1$ then σ is optimal in s for $Term(j)$.*
2. *If $Val^{Term(j)}(s) < 1$ then $\sup_\tau \mathbb{P}_s^{\tau,\pi}(Term(j)) < 1$.*

Moreover, σ is memoryless, and π only uses memory of size $|V|$. Such strategies can be computed in P-time for MDPs.

The proof goes along the lines of the proof of Theorem 1. It can be found in [2], together with an example that shows the memory use in π is necessary.

Similarly, both $Val^{Term(j)}(s) = 0$ and $Val^{Term(j)}(s) > 0$ are witnessed by pure and memoryless strategies for the respective players. Deciding which is the case is in P-time, by assigning the random states to player Max, obtaining a non-stochastic 2-player one-counter game, and using, e.g., [4, Theorem 12]. Finally, we note that from Proposition 13 and Lemma 14, it follows that:

▶ **Corollary 16.** *Given an SSG, $\mathcal{G}$, and reward function r, deciding whether the value of the termination objective $Term(j)$ equals 1 is at least as hard as Condon's [6]* quantitative *reachability problem, w.r.t. P-time many-one reductions.*

Acknowledgements

The work of Tomáš Brázdil has been supported by the Czech Science Foundation, grant No. P202/10/1469. The work of Václav Brožek has been supported by Newton International Fellowship from the Royal Society.

References

1 N. Berger, N. Kapur, L. J. Schulman, and V. Vazirani. Solvency Games. In *Proc. of FSTTCS'08*, 2008.
2 T. Brázdil, V. Brožek, and K. Etessami. One-Counter Stochastic Games. Technical Report abs/1009.5636, CoRR, http://arxiv.org/abs/1009.5636, 2010.
3 T. Brázdil, V. Brožek, K. Etessami, A. Kučera, and D. Wojtczak. One-Counter Markov Decision Processes. In *ACM-SIAM SODA*, pages 863–874, 2010. Full tech report: CoRR, abs/0904.2511, 2009. http://arxiv.org/abs/0904.2511.
4 T. Brázdil, P. Jančar, and A. Kučera. Reachability Games on Extended Vector Addition Systems with States. In *Proc. 37th Int. Coll. on Automata, Languages, and Programming (ICALP), Part II*, pages 478–489, 2010. Full tech report: FIMU-RS-2010-02, Faculty of Informatics, Masaryk University.
5 K. L. Chung. *A Course in Probability Theory.* Academic Press, 3rd edition, 2001.
6 A. Condon. The Complexity of Stochastic Games. *Inform. and Comput.*, 96:203–224, 1992.
7 C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events. *IEEE Trans. Automat. Control*, 43(10):1399–1418, 1998.
8 K. Etessami, D. Wojtczak, and M. Yannakakis. Quasi-birth-death processes, tree-like QBDs, probabilistic 1-counter automata, and pushdown systems. In *Proc. 5th Int. Symp. on Quantitative Evaluation of Systems (QEST)*, pages 243–253, 2008.
9 K. Etessami and M. Yannakakis. Recursive Markov decision processes and recursive stochastic games. In *Proc. 32nd ICALP*, pages 891–903, 2005.
10 K. Etessami and M. Yannakakis. Efficient qualitative analysis of classes of recursive Markov decision processes and simple stochastic games. In *Proc. of 23rd STACS'06*. Springer, 2006.
11 H. Gimbert. Pure stationary optimal strategies in markov decision processes. In *STACS*, pages 200–211, 2007.
12 H. Gimbert and F. Horn. Solving Simple Stochastic Tail Games. In *ACM-SIAM Symposium on Discrete Algorithms (SODA10)*, pages 847–862, 2010.
13 H. Gimbert and W. Zielonka. Pure and Stationary Optimal Strategies in Perfect-Information Stochastic Games. Technical Report hal-00438359, HAL, http://hal.archives-ouvertes.fr/hal-00438359/, 2009.
14 D. A. Martin. The Determinacy of Blackwell Games. *The Journal of Symbolic Logic*, 63(4):1565–1581, December 1998.
15 J. R. Norris. *Markov chains.* Cambridge University Press, 1998.
16 M. L. Puterman. *Markov Decision Processes.* J. Wiley and Sons, 1994.

ATL with Strategy Contexts: Expressiveness and Model Checking

Arnaud Da Costa[1], François Laroussinie[2], and Nicolas Markey[1]

1 **Lab. Spécification & Vérification, ENS Cachan & CNRS, France**
2 **LIAFA, Univ. Paris Diderot - Paris 7 & CNRS, France**

Abstract

We study the alternating-time temporal logics ATL and ATL* extended with *strategy contexts*: these make agents *commit to their strategies* during the evaluation of formulas, contrary to plain ATL and ATL* where strategy quantifiers *reset* previously selected strategies.

We illustrate the important expressive power of strategy contexts by proving that they make the extended logics, namely ATL$_{sc}$ and ATL$^\star_{sc}$, equally expressive: any formula in ATL$^\star_{sc}$ can be translated into an equivalent, linear-size ATL$_{sc}$ formula. Despite the high expressiveness of these logics, we prove that their model-checking problems remain decidable by designing a tree-automata-based algorithm for model-checking ATL$^\star_{sc}$ on the full class of n-player concurrent game structures.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.120

1 Introduction

Temporal logics and model checking. Thirty years ago, temporal logics (LTL, CTL) have been proposed for specifying properties of reactive systems, with the aim of automatically checking that those properties hold for these systems [18, 10, 19]. This *model-checking* approach to formal verification has been widely studied, with powerful algorithms and implementations, and successfully applied in many situations.

Alternating-time temporal logic (ATL). In the last ten years, temporal logics have been extended with the ability of specifying *controllability properties* of *multi-agent systems*: the evolution of a multi-agent system depends on the concurrent actions of several agents, and ATL extends CTL with *strategy quantifiers* [4]: it can express properties such as *agent A has a strategy to keep the system in a set of safe states, whatever the other agents do.*

Nesting strategy quantifiers. Assume that, in the formula above, "safe states" are those from which agent B has a strategy to reach her goal state q_B infinitely often, and consider the system depicted on Fig. 1, where the circled states are controlled by player A (meaning that Player A selects the transition to be fired from those state) and the square state is controlled by player B. It is easily seen that this game contains no "safe state": after each visit to q_B, Player A can decide to take the system to the rightmost state, from which q_B is not reachable. It follows that Player A has no strategy to keep the system in safe states.

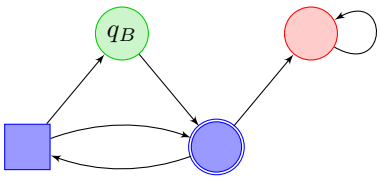


Figure 1 Example of a two-player turn-based game

Now, assume that Player A commits to always select the transition to the left, when the system is in the initial (double-circled) state. Then under this strategy, it suffices for

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 120–132
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Player B to always go to q_B when the system is in the square state in order to achieve her goal of visiting q_B infinitely often. The difference with the previous case is that here, Player B *takes advantage* of Player A's strategy in order to achieve her goal.

Both interpretations of our original property can make sense, depending on the context. However, the original semantics of ATL cannot capture the second interpretation: strategy quantifications in ATL "reset" previous strategies. While this is very convenient algorithmically (and makes ATL model-checking polynomial-time for some game models), it prevents ATL from expressing many interesting properties of games (especially non-zero-sum games).

In [7], we introduced an alternative semantics for ATL, where strategy quantifiers *store* strategies in a *context*. Those strategies then apply for evaluating the whole subformula, until they are explicitly removed from the context or replaced with a new strategy. We demonstrated the high expressiveness of this new semantics by showing that it can express important requirements, *e.g.* existence of equilibria or dominating strategies.

Our contribution. This work is a continuation of [7]. Our contribution in this paper is twofold: on the one hand, we prove that $\mathsf{ATL}^{\star}_{sc}$ is not more expressive than ATL_{sc}: this is a theoretical argument witnessing the expressive power of strategy contexts; it complements the more practical arguments presented in [7]. On the other hand, we develop an algorithm for $\mathsf{ATL}^{\star}_{sc}$ model-checking, based on alternating tree automata. Our algorithm uses a novel encoding of strategies into the execution tree of the underlying concurrent game structures. This way, it is valid for the whole class of concurrent game structures and without restrictions on strategies, contrary to previously existing algorithms on related extensions of ATL.

Related work. In the last three years, several approaches have been proposed to increase the expressiveness of ATL and $\mathsf{ATL}^{\star}$.

- *Strategy logic* [8, 9] extends LTL with first-order quantification over strategies. This allows for very expressive constructs: for instance, the property above would be written as $\exists\sigma_A.\ [\mathbf{G}\ (\exists\sigma_B.\ (\mathbf{G}\,\mathbf{F}\ q_B)\,(\sigma_A,\sigma_B))]\,(\sigma_A)$.
  This logic was only studied on two-player turn-based games in [8, 9], where a non-elementary algorithm is given. The algorithm we propose in this paper could be adapted to handle strategy logic in multi-player concurrent games.
- Q$D\mu$ [17] is a second-order extension of the propositional μ-calculus augmented with decision modalities. In terms of expressiveness, fixpoints allow for richer constructs than CTL- or LTL-based approaches. Again, model-checking has been proved to be decidable, but only over the class of alternating transition systems (as defined in [3]).
- *Stochastic game logic* [6] is an extension of ATL similar to ours, but in the stochastic case. It is proved undecidable in the general case, and decidable when strategy quantification is restricted to memoryless (randomized or deterministic) strategies.
- several other semantics of ATL, related to ours, are discussed in [1, 2]. A Δ_2^{P}-algorithm is proposed there for a subclass of our logic (where strategies stored in the context are *irrevocable* and cannot be overwritten), but no proof of correctness is given. In [20], an NP algorithm is proposed for the same subclass, but where strategy quantification is restricted to memoryless strategies.

By lack of space, some proofs are omitted in this paper, but they are detailed in [11].

2 ATL with strategy contexts

2.1 Concurrent game structures.

Concurrent game structures [4] are a multi-player extension of classical Kripke structures. Their definition is as follows:

▶ **Definition 1.** A *Concurrent Game Structure* (*CGS* for short) $\mathcal{C}$ is an 7-tuple $\langle \mathsf{Loc}, \mathsf{Lab}, \delta, \mathsf{Agt}, \mathcal{M}, \mathsf{Mov}, \mathsf{Edg}\rangle$ where:

- $\langle \mathsf{Loc}, \mathsf{Lab}, \delta\rangle$ is a finite Kripke structure, where Loc is the set of *locations*, $\mathsf{Lab}\colon \mathsf{Loc} \to 2^{\mathsf{AP}}$ is a labelling function, and $\delta \subseteq \mathsf{Loc} \times \mathsf{Loc}$ is the set of transitions;
- $\mathsf{Agt} = \{A_1, ..., A_p\}$ is a finite set of *agents* (or *players*);
- $\mathcal{M}$ is a finite, non-empty set of moves;
- $\mathsf{Mov}\colon \mathsf{Loc} \times \mathsf{Agt} \to \mathcal{P}(\mathcal{M}) \setminus \{\varnothing\}$ defines the (finite) set of possible moves of each agent in each location.
- $\mathsf{Edg}\colon \mathsf{Loc} \times \mathcal{M}^{\mathsf{Agt}} \to \delta$ is a transition table; with each location ℓ and each set of moves of the agents, it associates the resulting transition, which is required to depart from ℓ.

The size $|\mathcal{C}|$ of a CGS $\mathcal{C}$ is $|\mathsf{Loc}| + |\mathsf{Edg}|$, where $|\mathsf{Edg}|$ is the size of the transition table[1].

The intended behaviour of a CGS is as follows [4]: in a location ℓ, each player A_i in Agt chooses one among her possible moves m_i in $\mathsf{Mov}(\ell, A_i)$; the next transition to be fired is given by $\mathsf{Edg}(\ell, (m_1, ..., m_p))$. We write $\mathsf{Next}(\ell)$ for the set of all transitions corresponding to possible moves from ℓ, and $\mathsf{Next}(\ell, A_j, m_j)$, with $m_j \in \mathsf{Mov}(\ell, A_j)$, for the restriction of $\mathsf{Next}(\ell)$ to possible transitions from ℓ when player A_j plays the move m_j. We extend Mov and Next to coalitions (*i.e.*, sets of agents) in the natural way:

- given $A \subseteq \mathsf{Agt}$ and $\ell \in \mathsf{Loc}$, $\mathsf{Mov}(\ell, A)$ denotes the set of possible moves for coalition A from ℓ. Those moves m are composed of one single move per agent of the coalition, *i.e.*, $m = (m_a)_{a\in A}$.
- Given $m = (m_a)_{a\in A} \in \mathsf{Mov}(\ell, A)$, we let $\mathsf{Next}(\ell, A, m)$ denote the restriction of $\mathsf{Next}(\ell)$ to locations reachable from ℓ when every player $A_j \in A$ makes the move m_{A_j}.

A (finite or infinite) *path* of $\mathcal{C}$ is a sequence $\rho = \ell_0\ell_1 \dots$ of locations such that for any i, $\ell_{i+1} \in \mathsf{Next}(\ell_i)$. Finite paths are also called *history*. The length of a history $\rho = \ell_0\ell_1 \dots \ell_n$ is n. We write $\rho^{i\to j}$ for the part of ρ between ℓ_i and ℓ_j (inclusive). In particular, $\rho^{i\to j}$ is empty iff $j < i$. We simply write ρ^i for $\rho^{i\to i}$, denoting the $i+1$-st location ℓ_i of ρ. We also define $\mathsf{first}(\rho) = \rho^0$, and, if ρ has finite length n, $\mathsf{last}(\rho) = \rho^n$. Given a history π of length n and a path ρ s.t. $\mathsf{last}(\pi) = \mathsf{first}(\rho)$, the concatenation of π and ρ is the path $\tau = \pi \cdot \rho$ s.t. $\tau^{0\to n} = \pi$ and $\tau^{n\to\infty} = \rho$ (notice that the last location of π and the first location of ρ are "merged").

A *strategy* for a player $A_i \in \mathsf{Agt}$ is a function f_i that maps any history to a possible move for A_i, *i.e.*, satisfying $f_i(\ell_0 \dots \ell_m) \in \mathsf{Mov}(\ell_m, A_i)$. A strategy for a coalition A of agents is a mapping assigning a strategy to each agent in the coalition. The set of strategies for A is denoted $\mathsf{Strat}(A)$. The *domain* of $F_A \in \mathsf{Strat}(A)$ (denoted $\mathsf{dom}(F_A)$) is A. Given a coalition B, the strategy $(F_A)_{|B}$ (resp. $(F_A)_{\setminus B}$) denotes the restriction of F_A to the coalition $A \cap B$ (resp. $A \setminus B$).

Let ρ be a history of length n. A strategy $F_A = (f_j)_{A_j\in A}$ for some coalition A induces a set of paths from ρ, called the *outcomes* of F_A after (or from) ρ, and denoted $\mathsf{Out}(\rho, F_A)$:

[1] Our results would still hold (with the same complexity) if we consider symbolic CGSs [13], where the transition table is encoded succinctly as boolean formulas.

a path $\pi = \rho \cdot \ell_1\ell_2\dots$ is in $\mathsf{Out}(\rho, F_A)$ iff, writing $\ell_0 = \mathsf{last}(\rho)$, for all $i \geq 0$ there exists a set of moves $(m_k^i)_{A_k \in \mathsf{Agt}}$ such that $m_k^i \in \mathsf{Mov}(\ell_i, A_k)$ for all $A_k \in \mathsf{Agt}$, $m_k^i = f_{A_k}(\pi^{0\to n+i})$ if $A_k \in A$, and $\ell_{i+1} \in \mathsf{Next}(\ell_i, \mathsf{Agt}, (m_k^i)_{A_k\in\mathsf{Agt}})$. We write $\mathsf{Out}^\infty(\rho, F_A)$ for the set of infinite outcomes of F_A after ρ. Note that $\mathsf{Out}(\rho, F_A) \subseteq \mathsf{Out}(\rho, (F_A)_{|B})$ for any two coalitions A and B, and that $\mathsf{Out}(\rho, F_\varnothing)$ represents the set of all paths starting with ρ.

It is also possible to *combine* two strategies $F \in \mathsf{Strat}(A)$ and $F' \in \mathsf{Strat}(B)$, resulting in a strategy $F \circ F' \in \mathsf{Strat}(A \cup B)$ defined as follows: $(F \circ F')_{|A_j}(\rho)$ is $F_{|A_j}(\rho)$ (resp. $F'_{|A_j}(\rho)$) if $A_j \in A$ (resp. $A_j \in B \setminus A$).

Finally, given a strategy F and a history ρ, we define the strategy F^ρ corresponding to the behaviour of F after prefix ρ: it is defined, for any history π with $\mathsf{last}(\rho) = \mathsf{first}(\pi)$, as $F^\rho(\pi) = F(\rho \cdot \pi)$.

2.2 Alternating-time temporal logics.

The logics ATL and $\mathsf{ATL}^\star$ have been defined in [4] as extensions of CTL and $\mathsf{CTL}^\star$ with strategy quantification. Following [7], we further extend them with *strategy contexts*:

▶ **Definition 2.** The syntax of $\mathsf{ATL}^\star_{sc}$ is defined by the following grammar:

$$\begin{aligned}\mathsf{ATL}^\star_{sc} \ni \varphi_s, \psi_s &::= p \mid \neg\varphi_s \mid \varphi_s \vee \psi_s \mid \langle\!\cdot A \cdot\rangle\!\rangle\, \varphi_p \mid \rangle\!\cdot A \cdot\langle\!\rangle\, \varphi_s \\ \varphi_p, \psi_p &::= \varphi_s \mid \neg\varphi_p \mid \varphi_p \vee \psi_p \mid \mathbf{X}\,\varphi_p \mid \varphi_p\,\mathbf{U}\,\psi_p\end{aligned}$$

with $p \in \mathsf{AP}$ and $A \subseteq \mathsf{Agt}$. Formulas defined as φ_s are called *state-formulas*, while φ_p defines *path-formulas*. The logic ATL_{sc} is obtained by restricting the grammar of $\mathsf{ATL}^\star_{sc}$ path-formulas as follows:

$$\varphi_p, \psi_p ::= \neg\varphi_p \mid \mathbf{X}\,\varphi_s \mid \varphi_s\,\mathbf{U}\,\psi_s.$$

That a formula φ in $\mathsf{ATL}^\star_{sc}$ (or ATL_{sc}) holds (initially) along a computation ρ of a CGS $\mathcal{C}$ under a strategy context F (*i.e.*, a preselected strategy for some of the players, hence belonging to some $\mathsf{Strat}(A)$ for a coalition A), denoted $\mathcal{C}, \rho \models_F \varphi$, is defined as follows:

$$\begin{aligned}
\mathcal{C},\rho \models_F p &\quad\text{iff}\quad p \in \mathsf{Lab}(\mathsf{first}(\rho)) \\
\mathcal{C},\rho \models_F \neg\varphi &\quad\text{iff}\quad \mathcal{C},\rho \not\models_F \varphi \\
\mathcal{C},\rho \models_F \varphi \vee \psi &\quad\text{iff}\quad \mathcal{C},\rho \models_F \varphi \text{ or } \mathcal{C},\rho \models_F \psi \\
\mathcal{C},\rho \models_F \langle\!\cdot A \cdot\rangle\!\rangle\, \varphi_p &\quad\text{iff}\quad \exists F_A \in \mathsf{Strat}(A).\ \ \forall \rho' \in \mathsf{Out}^\infty(\mathsf{first}(\rho), F_A \circ F).\ \mathcal{C},\rho' \models_{F_A \circ F} \varphi_p \\
\mathcal{C},\rho \models_F \rangle\!\cdot A \cdot\langle\!\rangle\, \varphi_s &\quad\text{iff}\quad \mathcal{C},\rho \models_{F_{\setminus A}} \varphi_s \\
\mathcal{C},\rho \models_F \mathbf{X}\,\varphi_p &\quad\text{iff}\quad \mathcal{C},\rho^{1\to\infty} \models_{F^{\rho^{0\to 1}}} \varphi_p \\
\mathcal{C},\rho \models_F \varphi_p\,\mathbf{U}\,\psi_p &\quad\text{iff}\quad \exists i \geq 0.\ \mathcal{C},\rho^{i\to\infty} \models_{F^{\rho^{0\to i}}} \psi_p \text{ and } \forall 0 \leq j < i.\ \mathcal{C},\rho^{j\to\infty} \models_{F^{\rho^{0\to j}}} \varphi_p
\end{aligned}$$

We define the following shorthands, which will be useful in the sequel: $\top \stackrel{\text{def}}{=} p \vee \neg p$, $\bot \stackrel{\text{def}}{=} \neg\top$, $\mathbf{F}\,\varphi \stackrel{\text{def}}{=} \top\,\mathbf{U}\,\varphi$, $\mathbf{G}\,\varphi \stackrel{\text{def}}{=} \neg\mathbf{F}\neg\varphi$, $\langle\!\cdot A \cdot\rangle\!\rangle\, \varphi_s \stackrel{\text{def}}{=} \langle\!\cdot A \cdot\rangle\!\rangle\, (\bot\,\mathbf{U}\,\varphi_s)$ and $\langle\!\langle A \rangle\!\rangle\, \varphi \stackrel{\text{def}}{=} \rangle\!\mathsf{Agt}\langle\!\rangle\, \langle\!\cdot A \cdot\rangle\!\rangle\, \varphi$.

▶ **Example 3** (see [7] for more examples)**.** We illustrate the usefulness of strategy contexts with some examples. First, the last shorthand $\langle\!\langle A \rangle\!\rangle$ is the classical $\mathsf{ATL}^\star$ strategy quantifier (where each quantification resets the context), so that ATL_{sc} and $\mathsf{ATL}^\star_{sc}$ encompass ATL and $\mathsf{ATL}^\star$, respectively.

ATL_{sc} can also express qualitative equilibria properties, for instance Nash equilibria. Given the (non-zero-sum) objectives Φ_1 and Φ_2 of players 1 and 2, Nash equilibria are strategy profiles where none of the player can unilaterally improve her payoff. In other

terms, if the Player-1 strategy in the context is not winning against the Player-2 strategy, then there is no Player-1 winning strategy against this particular strategy of Player 2 (and symmetrically). Thus, the existence of a Nash equilibrium can be expressed as

$$\langle\!\cdot A_1, A_2 \cdot\!\rangle \Big[\big(\langle\!\cdot A_1 \cdot\!\rangle \, \Phi_1 \to \Phi_1 \big) \wedge \big(\langle\!\cdot A_2 \cdot\!\rangle \, \Phi_2 \to \Phi_2 \big) \Big]$$

As another example, we mention the interaction between a server S and different clients $(C_i)_i$, where we may want to express that the server can be programmed in such a way that each client C_i has a strategy to have its request granted. This could be written as

$$\langle\!\cdot S \cdot\!\rangle \, \mathbf{G} \Big[\bigwedge_i \big(\mathsf{req}_i \to \langle\!\cdot A_i \cdot\!\rangle \, \mathbf{F} \, \mathsf{grant}_i \big) \Big]$$

As stated in Lemma 4, the truth value of a *state* formula φ_s depends only on the strategy context F and the *first* state of the computation ρ where it is interpreted (thus we may simply write $\mathcal{C}, \mathsf{first}(\rho) \models_F \varphi_s$ when it raises no ambiguity).

▶ **Lemma 4.** *Let $\mathcal{C}$ be a CGS, and $F \in \mathsf{Strat}(A)$ be a strategy context. For any state formula φ_s, and for any two infinite paths ρ and ρ' with $\mathsf{first}(\rho) = \mathsf{first}(\rho')$, it holds*

$$\mathcal{C}, \rho \models_F \varphi_s \qquad \Leftrightarrow \qquad \mathcal{C}, \rho' \models_F \varphi_s.$$

Proof. The proof is by induction on the structure of φ_s: the result obviously holds for atomic propositions, and it is clearly preserved by boolean combinations and by the $\cdot\rangle A \langle\cdot$ operator. Finally, if $\varphi_s = \langle\!\cdot A \cdot\!\rangle \, \psi_s$, the result is immediate as the semantics only involves the first location of the path along which the formula is being evaluated. ◀

▶ Remark. Note that contrary to ATL, it is not possible to restrict to memoryless strategies (*i.e.*, that only depend on the current state) for ATL_{sc} formulas. For example, the formula $\langle\!\cdot A \cdot\!\rangle \, \mathbf{G} \, (\langle\!\cdot \varnothing \cdot\!\rangle \, \mathbf{F} \, P \wedge \langle\!\cdot \varnothing \cdot\!\rangle \, \mathbf{F} \, P')$ is equivalent in a standard Kripke struture (seen as a CGS with one single player A) to the $\mathsf{CTL}^\star$ formula $\mathsf{E}(\overset{\infty}{\mathbf{F}} P \wedge \overset{\infty}{\mathbf{F}} P')$ that may require strategies with memory. The next section provides more results on the extra expressiveness brought in by strategy contexts.

3 The expressive power of strategy contexts

As shown in [7], adding strategy contexts in formulas increases the expressive power of logics: ATL_{sc} (resp. $\mathsf{ATL}^\star_{sc}$) is strictly more expressive than ATL (resp. $\mathsf{ATL}^\star$). Game Logic (see [4]) can also be translated into $\mathsf{ATL}^\star_{sc}$ (while the converse is not true). In this section, we present some new results on the expressiveness of ATL_{sc}.

3.1 Alternating bisimulation.

Contrary to ATL, $\mathsf{ATL}^\star$, GL or AMC, our logics are not alternating-bisimulation invariant (see [5]), indeed we have:

▶ **Lemma 5.** *There exists two CGSs $\mathcal{C}$ and $\mathcal{C}'$, with an alternating-bisimulation linking two states ℓ_0 of $\mathcal{C}$ and ℓ'_0 in $\mathcal{C}'$, and an ATL_{sc} formula φ such that $\mathcal{C}, \ell_0 \models \varphi$ and $\mathcal{C}', \ell'_0 \not\models \varphi$.*

3.2 Relative expressiveness of ATL_{sc} and $\mathsf{ATL}^{\star}_{sc}$.

Surprisingly, strategy contexts bring ATL_{sc} to the same expressiveness as $\mathsf{ATL}^{\star}_{sc}$. This was already exemplified above, with the $\mathsf{CTL}^{\star}$ formula $\mathsf{E}(\overset{\infty}{\mathbf{F}}\, P \wedge \overset{\infty}{\mathbf{F}}\, P')$. We can extend this approach to any $\mathsf{ATL}^{\star}_{sc}$ formula: the idea is to

1. first use *full* strategy contexts (by adding universally quantified strategies) in order to be able to insert the $\langle\cdot\varnothing\cdot\rangle$ modality before every temporal modality, and
2. ensure that for every nested strategy quantifier $\langle\cdot A\cdot\rangle$, Coalition A cannot take advantage of the added strategies.

Given an $\mathsf{ATL}^{\star}_{sc}$ formula Φ and a coalition B, we define $\widehat{\Phi}^{[B]}$ inductively as follows:

$$\widehat{P}^{[B]} \stackrel{\text{def}}{=} P \qquad \widehat{\neg\varphi}^{[B]} \stackrel{\text{def}}{=} \neg\widehat{\varphi}^{[B]} \qquad \widehat{\varphi\wedge\psi}^{[B]} \stackrel{\text{def}}{=} \widehat{\varphi}^{[B]} \wedge \widehat{\psi}^{[B]}$$

$$\widehat{\mathbf{X}\,\varphi}^{[B]} \stackrel{\text{def}}{=} \langle\cdot\varnothing\cdot\rangle\,\mathbf{X}\,\widehat{\varphi}^{[B]} \qquad \widehat{\varphi\,\mathbf{U}\,\psi}^{[B]} \stackrel{\text{def}}{=} \langle\cdot\varnothing\cdot\rangle\,(\widehat{\varphi}^{[B]}\,\mathbf{U}\,\widehat{\psi}^{[B]})$$

$$\widehat{\langle\cdot A\cdot\rangle\,\varphi}^{[B]} \stackrel{\text{def}}{=} \langle\cdot A\cdot\rangle\,\neg\,\langle\cdot\mathsf{Agt}\backslash(A\cup B)\cdot\rangle\,\neg\widehat{\varphi}^{[A\cup B]} \qquad \widehat{\cdot\rangle A\langle\cdot\,\varphi}^{[B]} \stackrel{\text{def}}{=} \widehat{\varphi}^{[B\smallsetminus A]}$$

Clearly, $\widehat{\Phi}^{[B]}$ is an ATL_{sc} formula. The idea behind this translation is that a state-formula $\widehat{\varphi}^{A}$ interpreted in a strategy context F only depends on $F_{|A}$. We then have:

▶ **Lemma 6.** *Let $\mathcal{C}$ be a CGS, ℓ be one of its locations, and F be a strategy context. Then for any $\mathsf{ATL}^{\star}_{sc}$ formula φ, for any strategy context G s.t. $\mathsf{dom}(G) = \mathsf{Agt}\smallsetminus\mathsf{dom}(F)$, and for any outcome $\pi \in \mathsf{Out}^{\infty}(\ell, G\circ F)$, it holds: $\mathcal{C},\pi \models_F \varphi \;\Leftrightarrow\; \mathcal{C},\pi \models_{G\circ F} \widehat{\varphi}^{[\mathsf{dom}(F)]}$. Moreover, if φ is a state-formula, this result extends to any strategy context G s.t. $\mathsf{dom}(G)\cap\mathsf{dom}(F)=\varnothing$.*

Since our transformation does not depend on the underlying CGS, we get:

▶ **Theorem 7.** *Given a set of agents Agt, any $\mathsf{ATL}^{\star}_{sc}$ formula φ can be translated into an equivalent (under the empty context) ATL_{sc} formula $\widehat{\varphi}$ for any CGS based on Agt.*

Another consequence of the previous result is that any $\mathsf{ATL}^{\star}$ state-formula φ can be translated into the equivalent ATL_{sc} formula $\widehat{\varphi}^{\varnothing}$ in polynomial time. Thus we have:

▶ **Corollary 8.** *Model-checking ATL_{sc} is 2*EXPTIME*-hard.*

4 From ATL_{sc} to alternating tree automata

The main result of this section is the following:

▶ **Theorem 9.** *Model-checking ATL_{sc} formulas with at most k nested strategy quantifiers can be achieved in $(k+1)$*EXPTIME*. The* program complexity *(i.e., the complexity of model-checking a fixed ATL_{sc} formula) is in* EXPTIME*.*

The proof mainly consists in building an alternating tree automaton from a formula and a CGS. Similar approaches have already been proposed for strategy logic [9] or $\textsc{q}D\mu$ [17], but they were only valid for subclasses of CGSs: strategy logic was only studied on turn-based games, while the algorithm for $\textsc{q}D\mu$ was restricted to ATSs [3]. In both cases, the important point is that strategies are directly encoded as trees, with as many successors of a node as the number of possible moves from the corresponding node. With this representation, it is required that two different successors of a node correspond to two different states (which is the case for ATSs, hence for turn-based games): if this is not the case, the tree automaton may accept strategies that do not only depend on the sequence of states visited in the history,

but also on the sequence of moves proposed by the players. Our encoding is different: we work on the execution tree of the CGS under study, and label each node with possible moves of the players. We then have to focus on branches that correspond to outcomes of selected strategies, and check that they satisfy the requirement specified by the formula. Before presenting the detailed proof, we first introduce alternating tree automata and fix notations.

4.1 Trees and alternating tree automata

Let Σ and S be two finite sets. A Σ-*labelled* S-*tree* is a pair $\mathcal{T} = \langle T, l\rangle$, where

- $T \subseteq S^*$ is a non-empty set of finite words on S satisfying the following constraints: for any non-empty word $n = m \cdot s$ in T with $m \in T$ and $s \in S$, the word m is also in T;
- $l\colon T \to \Sigma$ is a labeling function.

Given such a tree $\mathcal{T} = \langle T, l\rangle$ and a node $n \in T$, the set of *directions from* n *in* T is the set $\mathsf{dir}_T(n) = \{s \in S \mid n{\cdot}s \in T\}$. The set of *successors of* n *in* T is $\mathsf{succ}_T(n) = \{n{\cdot}s \mid s \in \mathsf{dir}_T(n)\}$. We use $\mathcal{T}_n$ to denote the subtree rooted in n. An S-tree is complete if $T = S^*$, *i.e.*, if $\mathsf{dir}_T(n) = S$ for all $n \in T$. We may omit the subscript T when it is clear from the context.

The set of *infinite paths of* $\mathcal{T}$ is the set $\mathsf{Path}_{\mathcal{T}} = \{s_0{\cdot}s_1 \cdots \in S^\omega \mid \forall i \in \mathbb{N}.\ s_0{\cdot}s_1 \cdots s_i \in T\}$. Given such an infinite path $\pi = (s_i)_{i\in\mathbb{N}}$, we write $l(\pi)$ for the infinite sequence $(l(s_i))_{i\in\mathbb{N}} \in \Sigma^\omega$, and $\mathsf{Inf}(l(\pi))$ for the set of letters in Σ that appear infintely often along $l(\pi)$.

Assume that $\Sigma = \Sigma_1 \times \Sigma_2$, and pick a Σ-labelled S-tree $\mathcal{T} = \langle T, l\rangle$. For all $n \in T$, we write $l(n) = (l_1(n), l_2(n))$ with $l_i(n) \in \Sigma_i$ for $i \in \{1, 2\}$. Then for $i \in \{1, 2\}$, the *projection of* $\mathcal{T}$ *on* Σ_i, denoted by $\mathsf{proj}_{\Sigma_i}(\mathcal{T})$, is the Σ_i-labelled S-tree $\langle T, l_i\rangle$. Two Σ-labelled S-trees are Σ_i-*equivalent* if their projections on Σ_i are equal. These notions naturally extend to more complex alphabets, of the form $\prod_{i\in I} \Sigma_i$.

We now define alternating tree automata, which will be used in the proof. This requires the following definition: the set of *positive boolean formulas* over a finite set P of propositional variables is the set of formulas generated by: $\mathsf{PBF}(P) \ni \zeta ::= p \mid \zeta \wedge \zeta \mid \zeta \vee \zeta \mid \top \mid \bot$ where p ranges over P. That a valuation $v\colon P \to \{\top, \bot\}$ satisfies a formula in $\mathsf{PBF}(P)$ is defined in the natural way. We abusively say that a subset P' of P satisfies a formula $\varphi \in \mathsf{PBF}(P)$ iff the valuation $\mathbb{1}_{P'}$ (mapping the elements of P' to $\top$ and te elements of $P \setminus P'$ to $\bot$) satisfies φ. Since negation is not allowed, if $P' \models \varphi$ and $P' \subseteq P''$, then also $P'' \models \varphi$.

▶ **Definition 10.** Let S and Σ be two finite sets. An *alternating* S-*tree automaton on* Σ, or $\langle S, \Sigma\rangle$-ATA, is a 4-tuple $\mathcal{A} = \langle Q, q_0, \tau, \mathsf{Acc}\rangle$ where Q is a finite set of states, $q_0 \in Q$ is the initial state, Σ is a finite alphabet, $\tau\colon Q \times \Sigma \to \mathsf{PBF}(S \times Q)$ is the transition function, and $\mathsf{Acc}\colon Q^\omega \to \{\top, \bot\}$ is the acceptance function.

A *non-deterministic* S-*tree automaton on* Σ, or $\langle S, \Sigma\rangle$-NTA, is a $\langle S, \Sigma\rangle$-ATA in which conjunctions are not allowed for defining the transition function. The size of $\mathcal{A}$, denoted by $|\mathcal{A}|$, is the number of states in Q.

Let $\mathcal{A} = \langle Q, q_0, \tau, \mathsf{Acc}\rangle$ be an $\langle S, \Sigma\rangle$-ATA, and $\mathcal{T} = \langle T, l\rangle$ be a Σ-labelled S-tree. An *execution tree* of $\mathcal{A}$ on $\mathcal{T}$ is a $T \times Q$-labelled $S \times Q$-tree $\mathcal{E} = \langle E, p\rangle$ such that $p(\varepsilon) = (\varepsilon, q_0)$, and for each node $e \in E$ with $p(e) = (t, q)$, the set $\mathsf{dir}_E(e) = \{(s_0, q_0), (s_1, q_1), ..., (s_n, q_n)\} \subseteq S \times Q$ satisfies $\tau(q, l(t))$, and for all $0 \leq i \leq n$, the node $e{\cdot}(s_i, q_i)$ is labelled with $(t{\cdot}s_i, q_i)$. We write $p_S(e{\cdot}(s_i, q_i)) = t{\cdot}s_i$ and $p_Q(e{\cdot}(s_i, q_i)) = q_i$ for the two components of the labelling function.

An execution tree is *accepting* if $\mathsf{Acc}(p_Q(\pi)) = \top$ for any infinite path $\pi \in (S \times Q)^\omega$ in $\mathsf{Path}_{\mathcal{E}}$. A tree $\mathcal{T}$ is accepted by $\mathcal{A}$ iff there exists an accepting execution tree of $\mathcal{A}$ on $\mathcal{T}$. In the sequel, we use *parity acceptance condition*, given as a function $\Omega\colon Q \to \{0, ..., k-1\}$, from which Acc is defined as follows: $\mathsf{Acc}(p_Q(\pi)) = \top$ iff $\min\{\Omega(q) \mid q \in \mathsf{Inf}(p_Q(\pi))\}$ is

even. $\langle S, \Sigma\rangle$-ATAs with such accepting conditions are called $\langle S, \Sigma\rangle$-APTs, and given an $\langle S, \Sigma\rangle$-APT $\mathcal{A}$, the size of the image of Ω is called the *index* of $\mathcal{A}$, and is denoted by $\mathsf{idx}(\mathcal{A})$. Analogously, $\langle S, \Sigma\rangle$-NPTs are $\langle S, \Sigma\rangle$-NTAs with parity acceptance conditions.

4.2 Unwinding of a CGS

Let $\mathcal{C} = \langle \mathsf{Loc}, \mathsf{Lab}, \delta, \mathsf{Agt}, \mathcal{M}, \mathsf{Mov}, \mathsf{Edg}\rangle$ be an n-player CGS, where we assume w.l.o.g. that $\delta = \mathsf{Loc} \times \mathsf{Loc}$, and $\mathsf{Mov}(\ell, A_i) = \mathcal{M}$ for any state ℓ and any player A_i. Let ℓ_0 be a state of $\mathcal{C}$.

For each location $\ell \in \mathsf{Loc}$, we define $\Sigma(\ell) = \{\ell\} \times \{\mathsf{Lab}(\ell)\} \times \{\mathsf{Edg}(\ell)\}$, and $\Sigma^+(\ell) = \Sigma(\ell) \times (\mathcal{M} \cup \{\bot\})^{\mathsf{Agt}} \times 2^{\{p_o, p_l, p_r\}}$, where $\bot$ is a special symbol not in $\mathcal{M}$ and p_o, p_l and p_r are three fresh propositions not in AP. We let[2] $\Sigma_{\mathcal{C}} = \bigcup_{\ell\in\mathsf{Loc}} \Sigma(\ell)$, and $\Sigma^+_{\mathcal{C}} = \bigcup_{\ell\in\mathsf{Loc}} \Sigma^+(\ell)$.

The *unwinding* of $\mathcal{C}$ from ℓ_0 is the $\Sigma_{\mathcal{C}}$-labelled complete Loc-tree $\mathcal{U} = \langle U, v\rangle$ where $U = \mathsf{Loc}^*$ and $v(u) \in \Sigma(\mathsf{last}(\ell_0 \cdot u))$ for all $u \in U$. An *extended unwinding* of $\mathcal{C}$ from ℓ_0 is a $\Sigma^+_{\mathcal{C}}$-labelled complete Loc-tree $\mathcal{U}'$ such that $\mathsf{proj}_{\Sigma_{\mathcal{C}}}(\mathcal{U}') = \mathcal{U}$. For each letter σ of $\Sigma^+_{\mathcal{C}}$, we write σ_{Loc}, σ_{AP}, σ_{Edg}, σ_{str} and σ_p for the five components, and extend this subscripting notation for the labelling functions of trees (written l_{Loc}, l_{AP}, l_{Edg}, l_{str} and l_p).

In the sequel, we identify a node u of $\mathcal{U}$ (which is a finite word over Loc) with the finite path $\ell_0 \cdot u$ of $\mathcal{C}$. Notice that this sequence of states of $\mathcal{C}$ may correspond to no *real* path of $\mathcal{C}$, in case it involves a transition that is not in the image of Edg.

With $\mathcal{C}$ and ℓ_0, we associate a $\langle \mathsf{Loc}, \Sigma^+_{\mathcal{C}}\rangle$-APT $\mathcal{A}_{\mathcal{C},\ell_0} = \langle \overline{\mathsf{Loc}}, \overline{\ell_0}, \tau, \Omega\rangle$ s.t. $\overline{\mathsf{Loc}} = \{\overline{\ell} \mid \ell \in \mathsf{Loc}\}$, $\overline{\ell_0}$ is the initial state, Ω constantly equals 0 (hence any valid execution tree is accepting), and given a state $\overline{\ell} \in \overline{\mathsf{Loc}}$ and a letter $\sigma \in \Sigma^+_{\mathcal{C}}$, the transition function is defined as follows: if $\sigma \in \Sigma^+(\ell)$, we let $\tau(\overline{\ell}, \sigma) = \bigwedge_{\ell'\in\mathsf{Loc}}(\ell', \overline{\ell'})$, and otherwise, we let $\tau(\overline{\ell}, \sigma) = \bot$.

▶ **Lemma 11.** *Let $\mathcal{C}$ be a CGS and ℓ_0 be a state of $\mathcal{C}$. Let $\mathcal{T} = \langle T, l\rangle$ be a $\Sigma^+_{\mathcal{C}}$-labelled Loc-tree. Then $\mathcal{A}_{\mathcal{C},\ell_0}$ accepts $\mathcal{T}$ iff $\mathsf{proj}_{\Sigma_{\mathcal{C}}}(\mathcal{T})$ is the unwinding of $\mathcal{C}$ from ℓ_0.*

In the sequel, we also use automaton $\mathcal{A}_{\mathcal{C}}$, which accepts the union of all $\mathcal{L}(\mathcal{A}_{\mathcal{C},\ell_0})$ when ℓ_0 ranges over Loc. It is easy to come up with such an automaton, *e.g.* with Lemma 14 below.

4.3 Strategy quantification

Let $\mathcal{T} = \langle T, l\rangle$ be a $\Sigma^+_{\mathcal{C}}$-labelled complete Loc-tree accepted by $\mathcal{A}_{\mathcal{C},\ell_0}$. Such a tree defines *partial* strategies for each player: for $A \in \mathsf{Agt}$, and for each node $n \in T$, we define $\mathsf{strat}^{\mathcal{T}}_A(\ell_0 \cdot n) = l_{\mathsf{str}}(n)(A) \in \mathcal{M} \cup \{\bot\}$. For $D \subseteq \mathsf{Agt}$, we write $\mathsf{strat}^{\mathcal{T}}_D$ for $(\mathsf{strat}^{\mathcal{T}}_A)_{A\in D}$.

As a first step, for each $D \subseteq \mathsf{Agt}$, we build a $\langle \mathsf{Loc}, \Sigma^+_{\mathcal{C}}\rangle$-APT $\mathcal{A}_{\mathrm{strat}}(D)$ which will ensure that for all $A \in D$, $\mathsf{strat}^{\mathcal{T}}_A$ is *really* a strategy for player A, *i.e.*, never returns $\bot$. This automaton has only one state q_0, with $\tau(q_0, \sigma) = \bigwedge_{\ell\in\mathsf{Loc}}(\ell, q_0)$ provided that $\sigma_{\mathsf{str}}(A) \neq \bot$ for all $A \in D$. Otherwise, $\tau(q_0, \sigma) = \bot$. Finally, $\mathcal{A}_{\mathsf{strat}}$ accepts all trees having a valid execution tree (*i.e.*, Ω constantly equals 0). The following result is straightforward:

▶ **Lemma 12.** *Let $\mathcal{C}$ be a CGS, ℓ_0 be a location of $\mathcal{C}$, and $D \subseteq \mathsf{Agt}$. Let $\mathcal{T} = \langle T, l\rangle$ be a $\Sigma^+_{\mathcal{C}}$-labelled complete Loc-tree accepted by $\mathcal{A}_{\mathcal{C},\ell_0}$. Then $\mathcal{T}$ is accepted by $\mathcal{A}_{\mathsf{strat}}(D)$ iff for each player $A \in D$, $\mathsf{strat}^{\mathcal{T}}_A$ never equals $\bot$.*

We now build an automaton for checking that proposition p_o labels outcomes of $\mathcal{T}$. More precisely, let $D \subseteq \mathsf{Agt}$ be a set of players. The automaton $\mathcal{A}_{\mathsf{out}}(D)$ will accept $\mathcal{T}$ iff

[2] Notice that $|\Sigma_{\mathcal{C}}| = |\mathsf{Loc}|$ and $|\Sigma^+_{\mathcal{C}}|$ is linear in the size of the input, as we assume an explicit representation of the Edg function [13].

p_o labels exactly the outcomes of strategies $\mathsf{strat}_A^{\mathcal{T}}$ for players $A \in D$. This is achieved by the following two-state automaton $\mathcal{A}_{\mathsf{out}}(D) = \langle Q, q_\in, \tau, \Omega\rangle$: $Q = \{q_\in, q_\notin\}$, $q_\in$ is the initial state, Ω constantly equals 0, and the transition function is defined as follows: if $p_o \notin \sigma$, then $\tau(q_\in, \sigma) = \bot$ and $\tau(q_\notin, \sigma) = \bigwedge_{\ell\in\mathsf{Loc}}(\ell, q_\notin)$; otheriwse, $\tau(q_\notin, \sigma) = \bot$ and

$$\tau(q_\in, \sigma) = \bigwedge_{\ell\in\mathsf{Next}(\sigma,D)} (\ell, q_\in) \wedge \bigwedge_{\ell\notin\mathsf{Next}(\sigma,D)} (\ell, q_\notin)$$

where $\mathsf{Next}(\sigma, D)$ is

$$\{\ell \in \mathsf{Loc} \mid \exists (m_i)_i \in \mathcal{M}^{\mathsf{Agt}} \text{ s.t. } (\sigma_{\mathsf{Loc}}, \ell) = \sigma_{\mathsf{Edg}}(\sigma_{\mathsf{Loc}}, (m_i)_i) \text{ and } \forall A_i \in D.\ \sigma_{\mathsf{str}}(A_i) = m_i\}.$$

In other terms, $\mathsf{Next}(\sigma, D)$ returns the set of successor states of state σ_{Loc} if players in D follow the strategies given by σ_{str}, and according to the transition table σ_{Edg}. Notice that $\mathsf{Next}(\sigma, D)$ is non-empty iff $\sigma_{\mathsf{str}}(A_i) \neq \bot$ for all $A_i \in D$. We then have:

▶ **Lemma 13.** *Let $\mathcal{C}$ be a CGS, and ℓ_0 be one of its locations, and $D \subseteq \mathsf{Agt}$. Let $\mathcal{T} = \langle T, l\rangle$ be a $\Sigma_{\mathcal{C}}^+$-labelled complete Loc-tree accepted by $\mathcal{A}_{\mathcal{C},\ell_0}$ and $\mathcal{A}_{\mathsf{strat}}(D)$. Then $\mathcal{T}$ is accepted by $\mathcal{A}_{\mathsf{out}}(D)$ iff for all $n \in T$, $p_o \in l_p(n)$ iff the finite run $\ell_0 \cdot n$ is an outcome of $\mathsf{strat}_D^{\mathcal{T}}$ from ℓ_0.*

4.4 Boolean operations, projection, non-determinization, ...

In this section, we review some classical results about alternating tree automata, which we will use in our construction.

▶ **Lemma 14.** *[15, 16] Let $\mathcal{A}$ and $\mathcal{B}$ be two $\langle S, \Sigma\rangle$-APTs that respectively accept the languages A and B. We can build two $\langle S, \Sigma\rangle$-APTs $\mathcal{C}$ and $\mathcal{D}$ that respectively accept the languages $A \cap B$ and $\overline{A}$ (the complement of A in the set of Σ-labelled S-trees). The size and index of $\mathcal{C}$ are at most $(|\mathcal{A}| + |\mathcal{B}|)$ and $\max(\mathsf{idx}(\mathcal{A}), \mathsf{idx}(\mathcal{B})) + 1$, while those of $\mathcal{D}$ are $|A|$ and $\mathsf{idx}(\mathcal{A})$.*

▶ **Lemma 15.** *[16] Let $\mathcal{A}$ be a $\langle S, \Sigma\rangle$-APT. We can build a $\langle S, \Sigma\rangle$-NPT $\mathcal{N}$ accepting the same language as $\mathcal{A}$, and such that $|\mathcal{N}| \in 2^{O(|\mathcal{A}|\mathsf{idx}(\mathcal{A})\cdot\log(|\mathcal{A}|\mathsf{idx}(\mathcal{A})))}$ and $\mathsf{idx}(\mathcal{N}) \in O(|\mathcal{A}|\mathsf{idx}(\mathcal{A}))$.*

▶ **Lemma 16.** *[14] Let $\mathcal{A}$ be a $\langle S, \Sigma\rangle$-NPT, with $\Sigma = \Sigma_1 \times \Sigma_2$. For all $i \in \{1, 2\}$, we can build a $\langle S, \Sigma\rangle$-NPT $\mathcal{B}_i$ such that, for any tree $\mathcal{T}$, it holds: $\mathcal{T} \in \mathcal{L}(\mathcal{B}_i)$ iff $\exists \mathcal{T}' \in \mathcal{L}(\mathcal{A})$. $\mathsf{proj}_{\Sigma_i}(\mathcal{T}) = \mathsf{proj}_{\Sigma_i}(\mathcal{T}')$. The size and index of $\mathcal{B}_i$ are those of $\mathcal{A}$.*

▶ **Lemma 17.** *Let $\mathcal{A}$ be a $\langle S, \Sigma \times 2^{\{p\}}\rangle$-APT s.t. for any two $\Sigma \times 2^{\{p\}}$-labelled S-trees $\mathcal{T}$ and $\mathcal{T}'$ with $\mathsf{proj}_\Sigma(\mathcal{T}) = \mathsf{proj}_\Sigma(\mathcal{T}')$, we have $\mathcal{T} \in \mathcal{L}(\mathcal{A})$ iff $\mathcal{T}' \in \mathcal{L}(\mathcal{A})$. Then we can build a $\langle S, \Sigma \times 2^{\{p\}}\rangle$-APT $\mathcal{B}$ s.t. for all $\Sigma \times 2^{\{p\}}$-labelled S-tree $\mathcal{T} = \langle T, l\rangle$, it holds: $\mathcal{T} \in \mathcal{L}(\mathcal{B})$ iff $\forall n \in T$. $(p \in l(n)$ iff $\mathcal{T}_n \in \mathcal{L}(\mathcal{A}))$. Then $\mathcal{B}$ has size $O(|\mathcal{A}|)$ and index $\mathsf{idx}(\mathcal{A}) + 1$.*

4.5 Transforming an ATL$_{sc}$ formula into an alternating tree automaton

▶ **Lemma 18.** *Let $\mathcal{C}$ be a CGS with finite state space Loc. Let ψ be an ATL$_{sc}$-formula, and $D \subseteq \mathsf{Agt}$ be a coalition. We can build a $\langle \mathsf{Loc}, \Sigma_{\mathcal{C}}^+\rangle$-APT $\mathcal{A}_{\psi,D}$ s.t.*

- *for any $\Sigma_{\mathcal{C}}^+$-labelled complete Loc-tree $\mathcal{T}$ accepted by $\mathcal{A}_{\mathcal{C}}$ and by $\mathcal{A}_{\mathsf{strat}}(D)$, it holds*

$$\mathcal{T} \in \mathcal{L}(\mathcal{A}_{\psi,D}) \quad \Leftrightarrow \quad \mathcal{C}, l_{\mathsf{Loc}}(\varepsilon) \models_{\mathsf{strat}_D^{\mathcal{T}}} \psi;$$

- *for any two $\Sigma_{\mathcal{C}}^+$-labelled complete Loc-tree $\mathcal{T}$ and $\mathcal{T}'$ s.t. $\mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T}) = \mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T}')$, with $\Sigma'_{\mathcal{C}} = \Sigma_{\mathcal{C}} \times (\mathcal{M} \cup \{\bot\})^{\mathsf{Agt}}$, we have*

$$\mathcal{T} \in \mathcal{L}(\mathcal{A}_{\psi,D}) \quad \Leftrightarrow \quad \mathcal{T}' \in \mathcal{L}(\mathcal{A}_{\psi,D}).$$

The size of $\mathcal{A}_{\psi,D}$ is at most d-exponential, where d is the number of (nested) strategy quantifiers in ψ. Its index is $d-1$-exponential.

Sketch of proof. The proof proceeds by induction on the structure of formula ψ. The case of atomic propositions is straightforward. Applying Lemma 14, we immediately get the result for the case when φ is a boolean combination of subformulas.

We now sketch the proof for the case when $\psi = \langle\!\langle A\rangle\!\rangle \mathbf{X}\,\varphi$. The case formulas $\langle\!\langle A\rangle\!\rangle \varphi_1 \mathbf{U}\,\varphi_2$ and[3] $\langle\!\langle A\rangle\!\rangle \varphi_1 \mathbf{R}\,\varphi_2$ could be handled similarly. The idea of the construction is as follows: we use automaton $\mathcal{A}_{\mathsf{out}}(D\cup A)$ to label outcomes with p_o, $\mathcal{A}_{\varphi,D\cup A}$ to label nodes where φ holds, and build an intermediate automaton $\mathcal{A}_f$ to check that all the outcomes satisfy $\mathbf{X}\,\varphi$. We then project out the strategy of coalition A, which yields the automaton for $\langle\!\langle A\rangle\!\rangle \mathbf{X}\,\varphi$.

Assume that we have already built the automaton $\mathcal{A}_{\varphi,D\cup A}$ (inductively). Applying Lemma 17 to $\mathcal{A}_{\varphi,D\cup A}$ with the extra proposition p_r, we get an automaton $\mathcal{B}_{p_r,\varphi,D\cup A}$ such that, given a tree $\mathcal{T}=\langle T,l\rangle$ accepted by $\mathcal{A}_{\mathcal{C}}$ and $\mathcal{A}_{\mathsf{strat}}(D\cup A)$, it holds

$$\mathcal{T}\in\mathcal{L}(\mathcal{B}_{p_r,\varphi,D\cup A}) \quad\Leftrightarrow\quad \forall n\in T.\ (p_r\in l(n)\Leftrightarrow \mathcal{C}, l_{\mathsf{Loc}}(n)\models_{\mathsf{strat}^{\mathcal{T}_n}_{D\cup A}}\varphi). \tag{1}$$

In order to check that all the outcome satisfy $\mathbf{X}\,\varphi$, we simply have to build an automaton $\mathcal{A}_f$ for checking the $\mathsf{CTL}^\star$ property $\mathbf{A}(\mathbf{G}\,p_o\to\mathbf{X}\,p_r)$. We refer to [12] for this classical construction. This automaton $\mathcal{A}_f$ has the following property: for any $\Sigma^+_{\mathcal{C}}$-labelled Loc-tree $\mathcal{T}=\langle T,l\rangle$, we have

$$\mathcal{T}\in\mathcal{L}(\mathcal{A}_f) \quad\Leftrightarrow\quad \mathcal{T},\varepsilon\models \mathbf{A}(\mathbf{G}\,p_o\to\mathbf{X}\,p_r). \tag{2}$$

Now, let $\mathcal{H}$ be the product of $\mathcal{A}_{\mathsf{strat}}(A)$, $\mathcal{A}_{\mathsf{out}}(D\cup A)$, $\mathcal{A}_f$ and $\mathcal{B}_{p_r,\varphi,D\cup A}$, and let $\mathcal{T}$ be a tree accepted by $\mathcal{A}_{\mathcal{C}}$ and $\mathcal{A}_{\mathsf{strat}}(D)$. If $\mathcal{T}$ is accepted by $\mathcal{H}$, then $D\cup A\subseteq\mathsf{dom}(\mathcal{T})$ and all the outcomes of the strategy $\mathsf{strat}^{\mathcal{T}}_{D\cup A}$ from $l_{\mathsf{Loc}}(\varepsilon)$ satisfy $\mathbf{X}\,\varphi$.

The converse does not hold in general, but we prove a weaker form: from $\mathcal{T}=\langle T,l\rangle$, accepted by $\mathcal{A}_{\mathcal{C}}$ and $\mathcal{A}_{\mathsf{strat}}(D)$, and such that $D\cup A\subseteq\mathsf{dom}(\mathcal{T})$ and the outcomes of $\mathsf{strat}^{\mathcal{T}}_{D\cup A}$ from $l_{\mathsf{Loc}}(\varepsilon)$ satisfy $\mathbf{X}\,\varphi$, we build $\mathcal{T}'=\langle T,l'\rangle$ such that $\mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T})=\mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T}')$, and $\mathcal{T}'$ is accepted by $\mathcal{H}$. To do this, it suffices to modify the labelling of $\mathcal{T}$ with p_o and p_r, in such a way that $\mathcal{T}'$ is accepted by $\mathcal{A}_{\mathsf{out}}(D\cup A)$ and $\mathcal{B}_{p_r,\varphi,D\cup A}$. This ensures that $\mathcal{C}, l_{\mathsf{Loc}}(\varepsilon)\models_{\mathsf{strat}^{\mathcal{T}'}_{D\cup A}}\langle\!\langle\varnothing\rangle\!\rangle\mathbf{X}\,\varphi$, and that $\mathcal{T}'$ is also accepted by $\mathcal{A}_f$. In the end, we have that for any tree $\mathcal{T}=\langle T,l\rangle$ accepted by $\mathcal{A}_{\mathcal{C}}$ and $\mathcal{A}_{\mathsf{strat}}(D)$,

$$\begin{aligned} D\cup A\subseteq\mathsf{dom}(\mathcal{T}) \quad\text{and}\quad \mathcal{C}, l_{\mathsf{Loc}}(\varepsilon)\models_{\mathsf{strat}^{\mathcal{T}}_{D\cup A}}\langle\!\langle\varnothing\rangle\!\rangle\mathbf{X}\,\varphi \quad\Leftrightarrow\\ \exists\mathcal{T}'\text{ s.t. }\mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T}')=\mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T})\text{ and }T'\in\mathcal{L}(\mathcal{H}). \end{aligned} \tag{3}$$

Applying Lemma 15, we get a $\langle\Sigma^+_{\mathcal{C}},\mathsf{Loc}\rangle$-NPT $\mathcal{N}$ such that $\mathcal{L}(\mathcal{N})=\mathcal{L}(\mathcal{H})$. We can then apply Lemma 16 for $\Sigma^+_{\mathcal{C}}=(\Sigma_{\mathcal{C}}\times(\mathcal{M}\cup\{\bot\})^{\mathsf{Agt}\smallsetminus A})\times((\mathcal{M}\cup\{\bot\})^A\times 2^{p_o,p_l,p_r})$ on the NPT $\mathcal{N}$; the resulting $\langle\Sigma^+_{\mathcal{C}},\mathsf{Loc}\rangle$-NPT $\mathcal{P}$ accepts all trees $\mathcal{T}$ whose labelling on $(\mathcal{M}\cup\{\bot\})^A\times 2^{p_o,p_l,p_r}$ can be modified in order to have the tree accepted by $\mathcal{N}$. Then $\mathcal{P}$ satisfies both properties of the Lemma: the second property directly follows from the use Lemma 16. For the first one, pick $\mathcal{T}=\langle T,l\rangle$ accepted by $\mathcal{A}_{\mathcal{C}}$ and by $\mathcal{A}_{\mathsf{strat}}(D)$. If $\mathcal{T}$ is accepted by $\mathcal{P}$, then from Lemma 16, there exists a tree $\mathcal{T}'=\langle T,l'\rangle$, with the same labelling as $\mathcal{T}$ on $\Sigma_{\mathcal{C}}\times(\mathcal{M}\cup\{\bot\})^{\mathsf{Agt}\smallsetminus A})$, and accepted by $\mathcal{N}$. Since $\mathcal{L}(\mathcal{N})=\mathcal{L}(\mathcal{H})$, and from (3), we get that $D\cup A\subseteq\mathsf{dom}(\mathcal{T}')$ and

[3] The "release" modality $\mathbf{R}$ is the dual of $\mathbf{U}$, defined by $\varphi_1\mathbf{R}\,\varphi_2\equiv\neg[(\neg\varphi_1)\mathbf{U}\,(\neg\varphi_2)]$. Notice that $\mathbf{X}$ is self-dual as we only evaluate formulas along infinite outcomes.

$\mathcal{C}, l_{\mathsf{Loc}}(\varepsilon) \models_{\mathsf{strat}^{\mathcal{T}'}_{D\cup A}} \langle\!\langle \varnothing \rangle\!\rangle \mathbf{X}\,\varphi$. Thus $\mathsf{strat}^{\mathcal{T}'}_A$ is a strategy for coalition A, and it witnesses the fact that $\mathcal{C}, l_{\mathsf{Loc}}(\varepsilon) \models_{\mathsf{strat}^{\mathcal{T}'}_D} \langle\!\langle A \rangle\!\rangle \mathbf{X}\,\varphi$, and we get the desired result since $\mathsf{strat}^{\mathcal{T}}_D = \mathsf{strat}^{\mathcal{T}'}_D$. Conversely, if $\mathcal{C}, l_{\mathsf{Loc}}(\varepsilon) \models_{\mathsf{strat}^{\mathcal{T}}_D} \langle\!\langle A \rangle\!\rangle \mathbf{X}\,\varphi$, then we can modify the labelling of $\mathcal{T}$ with a witnessing strategy for A, obtaining a tree $\mathcal{T}'$ such that $\mathcal{C}, l_{\mathsf{Loc}}(\varepsilon) \models_{\mathsf{strat}^{\mathcal{T}'}_{D\cup A}} \langle\!\langle \varnothing \rangle\!\rangle \mathbf{X}\,\varphi$. From (3), $\mathcal{T}'$ can in turn be modified into a tree $\mathcal{T}''$, with $\mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T}'') = \mathsf{proj}_{\Sigma'_{\mathcal{C}}}(\mathcal{T}')$, in such a way that $\mathcal{T}'' \in \mathcal{L}(\mathcal{H})$. Finally, since the projections of $\mathcal{T}''$ and $\mathcal{T}$ coincide on $(\Sigma_{\mathcal{C}} \times (\mathcal{M} \cup \{\bot\})^{\mathsf{Agt}\setminus A})$, it holds that $\mathcal{T}$ is accepted by $\mathcal{P}$. This concludes the proof for $\langle\!\langle A \rangle\!\rangle \mathbf{X}\,\varphi$.

The proofs for the "until" and "release" modalities follow the same lines, using p_l and p_r as extra atmomic propositions for the left- and right-hand subformulas, and modifying automaton $\mathcal{A}_f$ so that it accepts trees satisfying $\mathbf{A}(\mathbf{G}\,p_o \to p_l\,\mathbf{U}\,p_r)$ and $\mathbf{A}(\mathbf{G}\,p_o \to p_l\,\mathbf{R}\,p_r)$, respectively. Finally, when $\psi = \rangle A \langle\, \varphi$, we let $\mathcal{A}_{\rangle A\langle\, \varphi, D} = \mathcal{A}_{\varphi, D\setminus A}$, which is easily proved to satisfy both requirements.

Unless $A = \varnothing$, the construction of the automaton for $\langle\!\langle A \rangle\!\rangle \mathbf{X}\,\varphi$ (or $\langle\!\langle A \rangle\!\rangle \varphi_1\,\mathbf{U}\,\varphi_2$ or $\langle\!\langle A \rangle\!\rangle \varphi_1\,\mathbf{R}\,\varphi_2$) involves an exponential blowup in the size and index of the automata for the subformulas, and the index is bilinear in the size and index of these automata. In the end, for a formula involving d nested non-empty strategy quantifiers, the automaton has size d-exponential and index $d-1$-exponential. ◀

▶ **Corollary 19.** *Given an ATL_{sc} formula φ, a CGS $\mathcal{C}$ and a state ℓ_0 of $\mathcal{C}$, we can built an alternating parity tree automaton $\mathcal{A}$ s.t. $\mathcal{L}(\mathcal{A}) \neq \varnothing$ iff $\mathcal{C}, \ell_0 \models_{\varnothing} \varphi$. Moreover, $\mathcal{A}$ has size d-exponential and index $d-1$-exponential, where d is the number of nested non-empty strategy quantifiers.*

Proof. It suffices to take the product of the automaton $\mathcal{A}_{\varphi,\varnothing}$ (from Lemma 18) with $\mathcal{A}_{\mathcal{C},\ell_0}$. In case this $\langle \mathsf{Loc}, \Sigma^+_{\mathcal{C}} \rangle$-APT accepts a tree $\mathcal{T}$, Lemma 18 entails that $\mathcal{C}, \ell_0 \models_{\varnothing} \varphi$. Conversely, if $\mathcal{C}, \ell_0 \models \varphi$, then the extended unwinding tree $\mathcal{T} = \langle T, l \rangle$ of $\mathcal{C}$ from ℓ_0 in which $l_{\mathsf{str}}(n) = \bot$ for all $n \in T$ is accepted by $\mathcal{A}_{\mathcal{C},\ell_0}$ (and, trivially, by $\mathcal{A}_{\mathsf{strat}}(\varnothing)$), and from Lemma 18, it is also accepted by $\mathcal{A}_{\varphi,\varnothing}$. ◀

Proof of Theorem 9. The first statement directly follows from the previous corollary, since emptiness of alternating parity tree automata $\mathcal{A}$ can be checked in time $\exp(O(|\mathcal{A}| \times \mathsf{idx}(\mathcal{A})))$.

For the second statement, notice that the size and index of $\mathcal{A}_{\varphi,\varnothing}$ in the proof of Corollary 19 do not depend on the CGS $\mathcal{C}$. Hence the automaton $\mathcal{A}$ of Corollary 19 has size linear in $|\mathcal{C}|$, and can be computed in time exponential in $|\mathcal{C}|$ (because $\mathcal{A}_{out}(D)$ requires the computation of $\mathsf{Next}(\sigma, D)$). Non-emptiness is then checked in time exponential in $|\mathcal{C}|$. ◀

▶ Remark. Our algorithm can easily be modified in order to handle $\mathsf{ATL}^{\star}_{sc}$. One solution is to rely on Theorem 7, but our translation from $\mathsf{ATL}^{\star}_{sc}$ to ATL_{sc} may double the number of nested non-empty strategy quantifiers. The algorithm would then be in $(2k+1)$-EXPTIME, where k is the number of nested strategy quantifications. Another solution is to adapt our construction, by replacing each state-subformula with a fresh atomic proposition, and build the automaton $\mathcal{A}_f$ for a more complex $\mathsf{CTL}^{\star}$ formula. This results in a $(k+1)$-EXPTIME algorithm. In both cases, the program complexity is unchanged, in EXPTIME.

Similarly, our algorithm could be modified to handle strategy logic [9]. One important difference is that strategy logic may require to store several strategies per player in the tree, while ATL_{sc} only stores one strategy per player. This would then be reflected in a modified version of the Next function we use when building $\mathcal{A}_{\mathsf{out}}(D)$, where we should also indicate which strategies we use for which player.

5 Conclusions

Strategy contexts provide a very expressive extension of the semantics of ATL, as we witnessed by the fact that ATL_{sc} and $\mathsf{ATL}^{\star}_{sc}$ are equally expressive. We also designed a tree-automata-based algorithm for model-checking both logics on the whole class of CGSs, based on a novel encoding of strategies as a tree.

Our algorithms involve a non-elementary blowup in the size of the formula, which we currently don't know if it can be avoided. Trying to establish lower-bounds on the complexity of the problems is part of our future works. Regarding expressiveness, ATL_{sc} can distinguish between alternating-bisimilar CGSs, and we are also looking for a behavioural equivalence that could characterize the distinguishing power of ATL_{sc}.

References

1 T. Ågotnes, V. Goranko, and W. Jamroga. Alternating-time temporal logics with irrevocable strategies. In *TARK'07*, p. 15–24, 2007.

2 T. Ågotnes, V. Goranko, and W. Jamroga. Strategic commitment and release in logics for multi-agent systems. Tech. Rep. 08-01, Clausthal U. of Technology, Germany, 2008.

3 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *COMPOS'97*, LNCS 1536, p. 23–60. Springer, 1998.

4 R. Alur, T. A. Henzinger, and O. Kupferman. Alternating-time temporal logic. *J. ACM*, 49(5):672–713, 2002.

5 R. Alur, T. A. Henzinger, O. Kupferman, and M. Y. Vardi. Alternating refinement relations. In *CONCUR'98*, LNCS 1466, p. 163–178. Springer, 1998.

6 Ch. Baier, T. Brázdil, M. Größer, and A. Kučera. Stochastic game logic. In *QEST'07*, p. 227–236. IEEE Comp. Soc., 2007.

7 Th. Brihaye, A. Da Costa, F. Laroussinie, and N. Markey. ATL with strategy contexts and bounded memory. In *LFCS'09*, LNCS 5407, p. 92–106. Springer, 2009.

8 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. In *CONCUR'07*, LNCS 4703, p. 59–73. Springer, 2007.

9 K. Chatterjee, T. A. Henzinger, and N. Piterman. Strategy logic. *Inf. & Comp.*, 208(6):677–693, 2010.

10 E. M. Clarke and E. A. Emerson. Design and synthesis of synchrization skeletons using branching-time temporal logic. In *LOP'81*, LNCS 131, p. 52–71. Springer, 1982.

11 A. Da Costa, F. Laroussinie, and N. Markey. Expressiveness and decidability of ATL with strategy contexts. Tech. Rep. LSV-10-14, Lab Spécification & Vérification, France, 2010.

12 O. Kupferman, M. Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model-checking. *J. ACM*, 47(2):312–360, 2000.

13 F. Laroussinie, N. Markey, and G. Oreiby. On the expressiveness and complexity of ATL. *LMCS*, 4(2), 2008.

14 D. E. Muller and P. E. Schupp. Alternating automata on infinite objects, determinacy and Rabin's theorem. In *EPIT'84*, LNCS 192, p. 99–107. Springer, 1985.

15 D. E. Muller and P. E. Schupp. Alternating automata on infinite trees. *TCS*, 54(2-3):267–276, 1987.

16 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. *TCS*, 141(1-2):69–107, 1995.

17 S. Pinchinat. A generic constructive solution for concurrent games with expressive constraints on strategies. In *ATVA'07*, LNCS 4762, p. 253–267. Springer, 2007.

18 A. Pnueli. The temporal logic of programs. In *FOCS'77*, p. 46–57. IEEE Comp. Soc., 1977.

19 J.-P. Queille and J. Sifakis. Specification and verification of concurrent systems in CESAR. In *SOP'82*, LNCS 137, p. 337–351. Springer, 1982.

20 D. Walther, W. van der Hoek, and M. Wooldridge. Alternating-time temporal logic with explicit strategies. In *TARK'07*, p. 269–278, 2007.

Reasoning About Strategies[§]

Fabio Mogavero[*†] [1], **Aniello Murano**[*] [1], **and Moshe Y. Vardi**[‡] [2]

1 **Universitá degli Studi di Napoli "Federico II", I-80126 Napoli, Italy.**
`{mogavero, murano}@na.infn.it`
2 **Rice University, Department of Computer Science, Houston, TX 77251-1892, U.S.A.**
`vardi@cs.rice.edu`

Abstract

In open systems verification, to formally check for reliability, one needs an appropriate formalism to model the interaction between open entities and express that the system is correct no matter how the environment behaves. An important contribution in this context is given by *modal logics for strategic ability*, in the setting of *multi-agent games*, such as ATL, ATL*, and the like. Recently, Chatterjee, Henzinger, and Piterman introduced *Strategy Logic*, which we denote here by $\mathrm{SL}_{\mathrm{CHP}}$, with the aim of getting a powerful framework for reasoning explicitly about strategies. $\mathrm{SL}_{\mathrm{CHP}}$ is obtained by using first-order quantifications over strategies and it has been investigated in the specific setting of two-agents turned-based game structures where a non-elementary model-checking algorithm has been provided. While $\mathrm{SL}_{\mathrm{CHP}}$ is a very expressive logic, we claim that it does not fully capture the strategic aspects of multi-agent systems.

In this paper, we introduce and study a more general strategy logic, denoted SL, for reasoning about strategies in multi-agent concurrent systems. We prove that SL strictly includes $\mathrm{SL}_{\mathrm{CHP}}$, while maintaining a decidable model-checking problem. Indeed, we show that it is 2EXPTIME-COMPLETE, thus not harder than that for ATL* and a remarkable improvement of the same problem for $\mathrm{SL}_{\mathrm{CHP}}$. We also consider the satisfiability problem and show that it is undecidable already for the sub-logic $\mathrm{SL}_{\mathrm{CHP}}$ under the concurrent game semantics.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.133

1 Introduction

In system design, *model checking* is a well-established formal method that allows to automatically check for global system correctness [4, 19, 5]. In such a framework, in order to check whether a system satisfies a required property, we express the system in a formal model (such as a *Kripke* structure), specify the property with a temporal-logic formula (such as LTL [18], CTL [4], or CTL* [7]), and check formally that the model satisfies the formula. In the last decade, interest has arisen in analyzing the behavior of individual components and sets of components in systems with several components. This interest has started in reactive systems, which are systems that interact continually with their environments. In *module checking* [14] the system is modeled as a module that interacts with its environment and correctness means that a desired property holds with respect to all such interactions.

Starting from the study of module checking, researchers have looked for logics focusing on strategic behavior of agents in multi-agent systems [1, 16, 10]. One of the most important development in this field is *Alternating-Time Temporal Logic* (ATL*, for short), introduced by Alur, Henzinger,

[§] Part of this research was done while the authors were visiting the Hebrew University.
[*] Work partially supported by MIUR PRIN Project n.2007-9E5KM8, ESF GAMES Project, and Vigevani Research Project Prize 2010.
[†] Part of this research was done while visiting the Rice University.
[‡] Work supported in part by NSF grants CCF-0613889, CCF-0728882, and CNS 1049862, by BSF grant 9800096, and by gift from Intel.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 133–144

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and Kupferman [1]. ATL* allows reasoning about strategies for agents with temporal goals. Formally, it is obtained as a generalization of CTL* in which the path quantifiers, "E" (*there exists*) and "A" (*for all*) are replaced with "*strategic modalities*" of the form $\langle\langle A \rangle\rangle$ and $[\![A]\!]$, where A is a set of *agents* (a.k.a. *players*). Strategic modalities over agent sets are used to express cooperation and competition among agents in order to achieve certain goals. In particular, these modalities express selective quantifications over those paths that are the results of infinite games between the coalition and its complement. ATL* formulas are interpreted over *game structures*, which model interacting processes. Given a game structure $\mathcal{S}$ and a set A of agents, the ATL* formula $\langle\langle A \rangle\rangle \psi$ is satisfied at a state s of $\mathcal{S}$ if there is a *strategy* for the agents in A such that, no matter the strategy that is executed by agents not in A, the resulting outcome of the interaction satisfies ψ at s. Thus, ATL* can express properties related to the interaction among agents, while CTL* can only express property of the global system. As an example, consider the property "processes α and β cooperate to ensure that a system (having more than two processes) never enters a fail state". This property can be be expressed by the ATL* formula $\langle\langle \{\alpha, \beta\} \rangle\rangle \mathsf{G} \neg \mathit{fail}$, where G is the classical temporal modality "*globally*". CTL*, in contrast, cannot express this property [1]. Indeed, CTL* can only say whether the set of all agents can or cannot prevent the system from entering a fail state. The price that one pays for the expressiveness of ATL* is increased complexity; both model checking and satisfiability checking are 2EXPTIME-COMPLETE [1, 20].

Despite its powerful expressiveness, ATL* suffers of the strong limitation that strategies are treated only implicitly, through modalities that refer to games between competing coalitions. To overcome this problem, Chatterjee, Henzinger, and Piterman introduced Strategy Logic ($\mathrm{SL}_{\mathrm{CHP}}$, for short) [3], a logic that treats strategies in two-player games as explicit first-order objects. In $\mathrm{SL}_{\mathrm{CHP}}$, the ATL* formula $\langle\langle \alpha \rangle\rangle \psi$ becomes $\exists x. \forall y. \psi(x, y)$, i.e., "there exists a player-α strategy x such that for all player-β strategies y, the unique infinite path resulting from the two players following the strategies x and y satisfies the property ψ". The explicit treatment of strategies in $\mathrm{SL}_{\mathrm{CHP}}$ allows to state many properties not expressible in ATL*. In particular, it is shown in [3] that ATL* corresponds to the proper one-alternation fragment of $\mathrm{SL}_{\mathrm{CHP}}$. Chatterjee et al. have shown that the model-checking problem for $\mathrm{SL}_{\mathrm{CHP}}$ is decidable, although only a non-elementary algorithm for it, both in the size of the system and the size formula, has been provided, leaving as open the question whether an algorithm with a better complexity exists or not. The question about the decidability of satisfiability checking for $\mathrm{SL}_{\mathrm{CHP}}$ was also left open in [3].

While the basic idea exploited in [3] to quantify over strategies, and thus to commit agent explicitly to certain strategies, turns out to be very powerful, as discussed above, the logic $\mathrm{SL}_{\mathrm{CHP}}$ introduced there has been defined and investigated only under the weak framework of two-players and turn-based games. Also, the specific syntax considered for $\mathrm{SL}_{\mathrm{CHP}}$ allows only a weak kind of strategy commitment. For example, $\mathrm{SL}_{\mathrm{CHP}}$ does not allow different players to share, in different contexts, the same strategy. These considerations, as well as all questions left open about $\mathrm{SL}_{\mathrm{CHP}}$, have led us to introduce and investigate a new *Strategy Logic*, denoted SL, as a more general framework than $\mathrm{SL}_{\mathrm{CHP}}$, for explicit reasoning about strategies in multi-player concurrent game structures. Syntactically, SL extends LTL by means of two *strategy quantifiers*, the existential $\langle\langle x \rangle\rangle$ and the universal $[\![x]\!]$, and an *agent binding* (α, x), where α is an agent and x is variable. Intuitively, these elements can be respectively read as *"there exists a strategy x"*, *"for all strategies x"*, and *"bind agent* α *to the strategy associated with x"*. For example, in a system with three agents α, β, γ, the previous ATL* formula $\langle\langle \{\alpha, \beta\} \rangle\rangle \mathsf{G} \neg \mathit{fail}$ can be expressed by the SL formula $\langle\langle x \rangle\rangle \langle\langle y \rangle\rangle [\![z]\!](\alpha, x)(\beta, y)(\gamma, z)(\mathsf{G} \neg \mathit{fail})$. The variables x and y are used to select two strategies for the agents α and β, respectively, and z is used to select all strategies for agent γ such that the composition of all these strategies results in a play where *fail* is never meet. Note that we can also require (by means of agent binding) that agents α and β share the same strategy, using the formula $\langle\langle x \rangle\rangle [\![z]\!](\alpha, x)(\beta, x)(\gamma, z)(\mathsf{G} \neg \mathit{fail})$. We can also

vary the structure of the game by changing the way the quantifiers alternate, for example, in the formula $\langle\langle x\rangle\rangle[[z]]\langle\langle y\rangle\rangle(\alpha,x)(\beta,y)(\gamma,z)(\mathsf{G}\,\neg \mathit{fail})$. In this case, x remains uniform w.r.t. z, but y becomes dependent on z. The last two examples show that SL is a proper extension of both ATL* and $\mathrm{SL}_{\mathrm{CHP}}$. It is worth noting that the pattern of modal quantifications over strategies and binding to agents can be extended to other logics than LTL, such as the linear μ-CALCULUS. In fact, the use of LTL here is only a matter of simplicity in presenting our framework, and changing the embedded temporal logic involves only few side-changes in the decision procedures.

As a main result in this paper, we show that the model-checking problem for SL is decidable and precisely PTIME in the size of the model and 2EXPTIME-COMPLETE in the size of the specification, thus not harder than that for ATL*. Remarkably, this result improves significantly the complexity of the model-checking problem for $\mathrm{SL}_{\mathrm{CHP}}$, for which only a non-elementary upper-bound was known [3]. The lower bound for the addressed problem immediately follows from ATL*, which SL includes. For the upper bound, we follow an *automata-theoretic approach* [13], by reducing the decision problem for the logic to the emptiness problem of automata. To this aim, we use *alternating Parity tree automata*, which are *alternating tree automata* (see [8], for a survey) along with a Parity acceptance condition [15]. Due to the exponential size of the required automaton and the EXPTIME complexity required for checking its emptiness, we get the desired 2EXPTIME upper bound.

As another important issue in this paper, we address the satisfiability problem for SL. By using a reduction from the *recurrent domino problem*, we show that this problem is highly undecidable, and in fact Σ^1_1-HARD, (i.e., it is not computably enumerable). Interestingly, the reduction we propose also holds for $\mathrm{SL}_{\mathrm{CHP}}$, under the concurrent game semantics. Thus, we show that in this setting also $\mathrm{SL}_{\mathrm{CHP}}$ is highly undecidable, while it remains an open question whether it is decidable or not in the turned-based framework. A key point to prove the undecidability of SL has been to show that this logic lacks of the bounded-tree model property, which does hold for ATL* [20].

Since the rise of temporal and modal program logics in the mid-to-late 1970s, we have learned to expect such logics to have a decidable satisfiability problem. In the context of temporal logic, decidability results were extended from LTL, to CTL* and to ATL*. SL deviates from this pattern. It has a decidable model-checking problem, but an undecidable satisfiability problem. In this, it is similar to first-order logic. The decidability of model checking for first-order logic is the foundation for query evaluation in relational databases, and undecidability of satisfiability is a challenge we need to contend with. At the same time, it is clear that SL has nontrivial fragments, for example, ATL*, which do have a decidable satisfiability problem. Identifying larger fragments of SL with a decidable satisfiability problem is an important research problem.

Related Work Several works have focused on extensions of ATL* to incorporate more powerful strategic constructs. Among them, we recall the logics *Alternating-Time μ-calculus* [1], *Game Logic* [1], $\mathrm{QD}\mu$ [17], and some extensions of ATL* considered in [2]. AMC and $\mathrm{QD}\mu$ are intrinsically different from SL (as well as $\mathrm{SL}_{\mathrm{CHP}}$ and ATL*) as they are obtained by extending the propositional μ-calculus [11] with strategic modalities. GL is strictly included in $\mathrm{SL}_{\mathrm{CHP}}$, but does not use any explicit treatment of strategies. Also the extensions of ATL* considered in [2] do not use any explicit treatment of strategies. Rather, they consider restrictions on the memory for strategy quantifiers. Thus, all the above logics are different from SL, which aims it at being a minimal but powerful logic to reason about strategic behavior in multi-agent systems.

Due to the lack of space, all proofs are omitted and reported in the full version.

2 Preliminaries

A *concurrent game structure* (CGS, for short) is a tuple $\mathcal{G} \triangleq \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, \lambda, \tau, s_0\rangle$, where AP and Ag are finite non-empty sets of *atomic propositions* and *agents*, Ac and St are enumerable non-empty

sets of *actions* and *states*, $s_0 \in \mathrm{St}$ is a designated *initial state*, and $\lambda : \mathrm{St} \to 2^{\mathrm{AP}}$ is a *labeling* function that maps each state to the set of atomic propositions true in that state. Let $\mathrm{Dc} \triangleq \mathrm{Ac}^{\mathrm{Ag}}$ be the set of *decisions*, commonly known as *action profiles*, that are functions from Ag to Ac representing the choices of an action for each agent. Then, $\tau : \mathrm{St} \times \mathrm{Dc} \to \mathrm{St}$ is a *transition* function mapping a state and a decision to a state. Intuitively, CGSs provide a generalization of *labeled transition systems*, modeling multi-agent systems, viewed as *multi-player games* in which players perform *concurrent actions*, chosen strategically as a function of the history of the game. Note that elements in St are not global states of the system, but states of the environment in which the agents operate. Thus, they can be viewed as states of the game, which do not include the local states of the agents. By $|\mathcal{G}| \triangleq |\mathrm{St}| \cdot |\mathrm{Dc}|$ we denote the *size* of $\mathcal{G}$, which also corresponds to the size $|\mathsf{dom}(\tau)|$ of the transition function τ. If the set of actions is finite, i.e., $b = |\mathrm{Ac}| < \infty$, we say that $\mathcal{G}$ is *b-bounded*, or simply *bounded*. If both the sets of actions and states are finite, we say that $\mathcal{G}$ is *finite*. It is immediate to note that $\mathcal{G}$ is finite iff it has a finite size.

A *track* (resp., *path*) is a finite (resp., an infinite) sequence of states $\rho \in \mathrm{St}^*$ (resp., $\pi \in \mathrm{St}^\omega$) such that, for all $0 \leq i < |\rho| - 1$ (resp., $i \in \mathbb{N}$), there exists $\mathsf{d} \in \mathrm{Dc}$ such that $\rho_{i+1} = \tau(\rho_i, \mathsf{d})$ (resp., $\pi_{i+1} = \tau(\pi_i, \mathsf{d})$). Intuitively, tracks and paths of a CGS $\mathcal{G}$ are legal sequences of reachable states in $\mathcal{G}$ that can be seen as a description of the possible *outcomes* of the game modeled by $\mathcal{G}$. A track ρ is said *non-trivial* iff $|\rho| > 0$. We use $\mathrm{Trk} \subseteq \mathrm{St}^+$ (resp., $\mathrm{Pth} \subseteq \mathrm{St}^\omega$) to indicate the sets of all non-trivial tracks (resp., paths). By $\mathsf{fst}(\rho) \triangleq \rho_0$ (resp., $\mathsf{lst}(\rho) \triangleq \rho_{|\rho|-1}$), we denote the *first* (resp., *last*) state of the track ρ and, by $\rho_{\leq i}$, we denote the *prefix* up to the state of index $i < |\rho|$ of the track ρ, i.e., the track built by the first $i+1$ states $\rho_0, \ldots, \rho_i$ of ρ. The notations of first and prefix apply also to paths.

A *strategy* is a partial function $\mathsf{f} : \mathrm{Trk} \rightharpoonup \mathrm{Ac}$, non associated to any particular agent, mapping each non-trivial track in its domain to an action. Intuitively, a strategy is a *plan* for an agent that contains all choices of moves as a function of the history of the current outcome. We use Str to indicate the sets of all strategies. For a state s, we say that f is *s-total* iff it is defined on all non-trivial tracks starting in s, i.e., $\mathsf{dom}(\mathsf{f}) = \{\rho \in \mathrm{Trk} \mid \mathsf{fst}(\rho) = s\}$. For a track $\rho \in \mathsf{dom}(\mathsf{f})$, by f_ρ we denote the *translation* of f along ρ, i.e., the $\mathsf{lst}(\rho)$-total strategy such that $\mathsf{f}_\rho(\mathsf{lst}(\rho) \cdot \rho') = \mathsf{f}(\rho \cdot \rho')$, for all $\mathsf{lst}(\rho) \cdot \rho' \in \mathsf{dom}(\mathsf{f}_\rho)$.

Let Var be a fixed set of *variables*. An *assignment* is a partial function $\chi : \mathrm{Ag} \cup \mathrm{Var} \rightharpoonup \mathrm{Str}$ mapping every agent and variable to a strategy. An assignment χ is *complete* iff $\mathrm{Ag} \subseteq \mathsf{dom}(\chi)$. We use Asg to indicate the sets of all assignments. For a state s, we say that χ is *s-total* iff all strategies $\chi(l)$ are s-total too, for $l \in \mathsf{dom}(\chi)$. Let ρ be a track and χ be an $\mathsf{fst}(\rho)$-total assignment. By χ_ρ we denote the *translation* of χ along ρ, i.e., the $\mathsf{lst}(\rho)$-total assignment with $\mathsf{dom}(\chi_\rho) = \mathsf{dom}(\chi)$, such that $\chi_\rho(l) = \chi(l)_\rho$, for all $l \in \mathsf{dom}(\chi)$. Intuitively, the translation χ_ρ is the update of all strategies contained into the assignment χ, after the history of the game becomes ρ. Let χ be an assignment, a be an agent, x be a variable, and f be a strategy. Then, by $\chi[a \mapsto \mathsf{f}]$ and $\chi[x \mapsto \mathsf{f}]$ we denote, respectively, the new assignments defined on $\mathsf{dom}(\chi) \cup \{a\}$ and $\mathsf{dom}(\chi) \cup \{x\}$ that return f on a and x and are equal to χ on the remaining part of its domain. Note that, if χ and f are s-total, $\chi[a \mapsto \mathsf{f}]$ and $\chi[a \mapsto \mathsf{f}]$ are s-total, too.

Finally, a path π starting in a state s is a *play* w.r.t. a complete s-total assignment χ ((χ, s)-*play*, for short) iff, for all $i \in \mathbb{N}$, it holds that $\pi_{i+1} = \tau(\pi_i, \mathsf{d})$, where $\mathsf{d}(a) = \chi(a)(\pi_{\leq i})$, for all $a \in \mathrm{Ag}$. Note that there is a unique (χ, s)-play. Intuitively, a play is the outcome of the game determined by all the agent strategies participating to the game.

In the sequel of the paper, we use the Greek letters "α, β, γ" with indexes to indicate specific agents of a CGS, while we use the Latin letter "a" as a meta-variable on the agents themselves.

3 Strategy Logic

In this section, we formally introduce SL and discuss its main properties. In particular, we show that it does not have the bounded-tree model property.

Syntax. SL syntactically extends LTL by means of two *strategy quantifiers*, the existential $\langle\!\langle x \rangle\!\rangle$ and the universal $[\![x]\!]$, and an *agent binding* (a,x), where a is an agent and x is a variable. Intuitively, these new elements can be read, respectively, as *"there exists a strategy x"*, *"for all strategies x"*, and *"bind agent a to the strategy associated with variable x"*. The formal syntax of SL follows.

▶ **Definition 3.1** (Syntax)**.** SL *formulas* are built from the sets of atomic propositions AP, variables Var, and agents Ag, in the following way, where $p \in \mathrm{AP}$, $x \in \mathrm{Var}$, and $a \in \mathrm{Ag}$:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{X}\varphi \mid \varphi \mathsf{U} \varphi \mid \varphi \mathsf{R} \varphi \mid \langle\!\langle x \rangle\!\rangle \varphi \mid [\![x]\!]\varphi \mid (a,x)\varphi.$$

We now introduce some auxiliary syntactical notation for the definition of the semantics. By $\mathsf{free}(\varphi)$ we denote the set of *free agents/variables* of φ defined as the subset of $\mathrm{Ag} \cup \mathrm{Var}$ containing *(i)* all the agents for which there is no variable application after the occurrence of a temporal operator and *(ii)* all the variables for which there is an application but no quantifications. For example, let $\varphi = \langle\!\langle x \rangle\!\rangle (\alpha,x)(\beta,y)(\mathsf{F}\, p)$ be the formula on agents $\mathrm{Ag} = \{\alpha,\beta,\gamma\}$. Then, we have $\mathsf{free}(\varphi) = \{\gamma,y\}$, since γ is an agent without any application after $\mathsf{F}\, p$ and y has no quantification at all. A formula φ without free agents (resp., variables), i.e., with $\mathsf{free}(\varphi) \cap \mathrm{Ag} = \emptyset$ (resp., $\mathsf{free}(\varphi) \cap \mathrm{Var} = \emptyset$), is named *agent-closed* (resp., *variable-closed*). If φ is both agent- and variable-closed, it is named *sentence*.

Semantics. As for ATL*, we define the semantics of SL w.r.t. concurrent game structures. For a CGS $\mathcal{G}$, a state s, and an s-total assignment χ with $\mathsf{free}(\varphi) \subseteq \mathsf{dom}(\chi)$, we write $\mathcal{G},\chi,s \models \varphi$ to indicate that the formula φ holds at s under the assignment χ. Similarly, if χ is a complete assignment, for the (χ,s)-play π and a natural number k, we write $\mathcal{G},\chi,\pi,k \models \varphi$ to indicate that φ holds at the position k of π. The semantics of the SL formulas involving p, $\neg$, $\wedge$, and $\vee$, as well as that for the temporal operators X, U, and R, is defined as usual in LTL and we omit it here (see [13], for a survey). The semantics of the remaining part, which involves quantifications and bindings, follows.

▶ **Definition 3.2** (Semantics)**.** Given a CGS $\mathcal{G} = \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, \lambda, \tau, s_0 \rangle$, an SL formula φ, a variable $x \in \mathrm{Var}$, a state $s \in \mathrm{St}$, and an s-total assignment χ with $\mathsf{free}(\varphi) \subseteq \mathsf{dom}(\chi) \cup \{x\}$, it holds that:

1. $\mathcal{G},\chi,s \models \langle\!\langle x \rangle\!\rangle \varphi$ iff there is an s-total strategy f such that $\mathcal{G},\chi[x \mapsto \mathsf{f}],s \models \varphi$;
2. $\mathcal{G},\chi,s \models [\![x]\!]\varphi$ iff for all s-total strategies f it holds that $\mathcal{G},\chi[x \mapsto \mathsf{f}],s \models \varphi$.

Moreover, if $\mathsf{free}(\varphi) \cup \{x\} \subseteq \mathsf{dom}(\chi) \cup \{a\}$ for an agent $a \in \mathrm{Ag}$, it holds that:

3. $\mathcal{G},\chi,s \models (a,x)\varphi$ iff $\mathcal{G},\chi[a \mapsto \chi(x)],s \models \varphi$.

Finally, if χ is also complete, where π is the (χ,s)-play and $k \in \mathbb{N}$, it holds that:

4. $\mathcal{G},\chi,s \models \varphi$ iff $\mathcal{G},\chi,\pi,0 \models \varphi$;
5. $\mathcal{G},\chi,\pi,k \models \varphi$ iff $\mathcal{G},\chi_{\pi_{\leq k}},\pi_k \models \varphi$.

Intuitively, at Items 1 and 2, respectively, we evaluate existential and universal quantifiers over strategies. At Item 3, by means of an agent binding (a,x), we commit the agent a to a strategy contained in the variable x. Finally, Items 4 and 5 can be easily understood by looking at their analogous path and state formulas in ATL*. In fact, Item 4 can be viewed as the rule that allows to move the verification process from states to paths and, vice versa, Item 5 from paths to states.

A CGS $\mathcal{G}$ is a *model* of an SL sentence φ, denoted by $\mathcal{G} \models \varphi$, iff $\mathcal{G},\varnothing,s_0 \models \varphi$, where $\varnothing$ is the empty assignment. Moreover, φ is *satisfiable* iff there is a model for it. For two SL formulas φ_1 and φ_2 we say that φ_1 is *equivalent* to φ_2, formally $\varphi_1 \equiv \varphi_2$, iff, for all CGSs $\mathcal{G}$, states s, and s-defined assignments χ with $\mathsf{free}(\varphi_1) \cup \mathsf{free}(\varphi_2) \subseteq \mathsf{dom}(\chi)$, it holds that $\mathcal{G},\chi,s \models \varphi_1$ iff $\mathcal{G},\chi,s \models \varphi_2$.

As an example, let $\varphi = \langle\!\langle x \rangle\!\rangle [\![y]\!] \langle\!\langle z \rangle\!\rangle (\alpha,x)(\beta,y)(\mathsf{X}\, p) \wedge (\alpha,y)(\beta,z)(\mathsf{X}\, q)$. First, note that α and β both use the strategy associated with y to achieve the goals $\mathsf{X}\, q$ and $\mathsf{X}\, p$, respectively. A model for φ is $\mathcal{G} = \langle \{p,q\}, \{\alpha,\beta\}, \{0,1\}, \{s_0,s_1,s_2,s_3\}, \lambda, \tau, s_0 \rangle$, where $\lambda(s_0) = \emptyset$, $\lambda(s_1) = \{p\}$, $\lambda(s_2) = \{p,q\}$, $\lambda(s_3) = \{q\}$, $\tau(s_0,(0,0)) = s_1$, $\tau(s_0,(0,1)) = s_2$, $\tau(s_0,(1,0)) = s_3$, and all the remaining transitions (with any action) go to s_0. Clearly, $\mathcal{G},s_0 \models \varphi$ by letting, on s_0, x to chose action 0 (the goal $\mathsf{X}\, p$ is

satisfied for any choice of y, since we can move from s_0 to s_1 or s_2, both labeled with p) and z to choose action 1 when y has action 0 and, vice versa, z to 0 when y has 1 (in both the cases, the goal $\mathsf{X}\,q$ is satisfied, since one can move from s_0 to s_2 or s_3, both labeled with q).

An important property that is possible to express in SL, but neither in ATL* nor in $\mathrm{SL}_{\mathrm{CHP}}$, is the existence of *deterministic multi-player Nash equilibria*. For example, consider n agents $\alpha_1,\ldots,\alpha_n$ each of them having the LTL goals $\psi_1,\ldots,\psi_n$. Then, we can express the existence of a *strategy profile* $(x_1,\ldots,x_n)$ that is a Nash equilibrium for $\alpha_1,\ldots,\alpha_n$ w.r.t. $\psi_1,\ldots,\psi_n$ by using the sentence $\langle\!\langle x_1\rangle\!\rangle\cdots\langle\!\langle x_n\rangle\!\rangle(\alpha_1,x_1)\cdots(\alpha_n,x_n)(\bigwedge_{i=1}^{n}(\langle\!\langle y\rangle\!\rangle(\alpha_i,y)\psi_i)\rightarrow\psi_i)$. Informally, this sentence asserts that every agent has the "best" strategy once all the strategies of the remaining agents have been fixed. Note that here we have only considered equilibria under deterministic strategies.

Basic properties. We now investigate some basic properties of SL that turn out to be important for their own and useful to prove the decidability of the model checking problem and the undecidability of the satisfiability one. In particular, for the introduced logic we investigate the tree and finite model properties. To this aim, we define a generalization to CGS of the classical concept of unwinding of labeled transition systems, which allows us to show that SL has the (general) tree model property, but neither the bounded-tree model property nor the finite model property. As preliminary, we need to formally state the concepts of *concurrent game tree* and *unwinding* of a game structure.

A *concurrent game tree* (CGT, for short) is a CGS $\mathcal{U}=\langle \mathrm{AP},\mathrm{Ag},\mathrm{Ac},\mathrm{St},\lambda,\tau,\varepsilon\rangle$, where $\mathrm{St}\subseteq\Delta^*$ is a tree for a given set of directions Δ and $s\cdot t\in\mathrm{St}$ iff there is a decision $\mathsf{d}\in\mathrm{Dc}$ such that $\tau(s,\mathsf{d})=s\cdot t$, for all $s\in\mathrm{St}$ and $t\in\Delta$. For a CGS $\mathcal{G}=\langle \mathrm{AP},\mathrm{Ag},\mathrm{Ac},\mathrm{St},\lambda,\tau,s_0\rangle$, the *unwinding* of $\mathcal{G}$ is the CGT $\mathcal{G}^{\mathcal{U}}\triangleq\langle \mathrm{AP},\mathrm{Ag},\mathrm{Ac},\mathrm{St}',\lambda',\tau',\varepsilon\rangle$, where St is the set of directions of the tree, the states in $\mathrm{St}'=\{\rho\in\mathrm{St}^* \mid s_0\cdot\rho\in\mathrm{Trk}\}$ are the suffixes of the tracks starting in s_0, and, for all $s\in\mathrm{St}'$, $\mathsf{d}\in\mathrm{Dc}$, and $t\in\mathrm{St}$, it holds that $\tau'(s,\mathsf{d})=s\cdot\tau(\mathsf{lst}(s),\mathsf{d})$ and there is a surjective function $\mathsf{unw}:\mathrm{St}'\rightarrow\mathrm{St}$ such that *(i)* $\mathsf{unw}(\varepsilon)=s_0$, *(ii)* $\mathsf{unw}(s\cdot t)=t$, and *(iii)* $\lambda'(s)=\lambda(\mathsf{unw}(s))$.

We say that SL has the *tree model property* if every satisfiable formula is satisfiable by a CGT. Actually, next theorem reports that SL is invariant under unwinding, so, it has the tree model property. This can be shown by using a proof by induction on the structure of the formula, making use of the unwinding technique defined above.

▶ **Theorem 3.3.** *$\mathcal{G}\models\varphi$ iff $\mathcal{G}^{\mathcal{U}}\models\varphi$.*

We now move to the negative results about SL, namely, that it neither has the finite model property nor the bounded-tree model property. We recall that a modal logic has the bounded-tree model property (resp., finite model property) if whenever a formula is satisfiable, it is so on a model having a tree shape (resp., a finite number of states) in which every state has at most n successors, for a natural number n. Clearly, if a modal logic with the tree model property has the finite model property, it has the bounded-tree model property as well. The other direction may not hold, instead. To prove both results, we introduce in Definition 3.4 the formula φ^{ord} to be used as a counterexample.

▶ **Definition 3.4.** Let $x_1<x_2\triangleq\langle\!\langle y\rangle\!\rangle\,\varphi^{in}(x_1,x_2,y)$, where $\varphi^{in}(x_1,x_2,y)\triangleq(\beta,y)((\alpha,x_1)(\mathsf{X}\,p)\wedge(\alpha,x_2)(\mathsf{X}\neg p))$ is an agent-closed SL formula, named *partial order*, on the sets $\mathrm{AP}=\{p\}$ and $\mathrm{Ag}=\{\alpha,\beta\}$. Then, the SL *order sentence* $\varphi^{ord}\triangleq\varphi^{unb}\wedge\varphi^{trn}$ is the conjunction of the following two sentences, called *strategy unboundedness* and *strategy transitivity* requirements:

1. $\varphi^{unb}\triangleq[\![x_1]\!]\langle\!\langle x_2\rangle\!\rangle\, x_1<x_2$;
2. $\varphi^{trn}\triangleq[\![x_1]\!][\![x_2]\!][\![x_3]\!]\,(x_1<x_2\wedge x_2<x_3)\rightarrow x_1<x_3$.

Intuitively, φ^{unb} asserts that, for each strategy x_1, there is a different strategy x_2 in relation of $<$ w.r.t. the first one, i.e., $<$ has no upper bound, while φ^{trn} expresses the fact that the relation $<$ is transitive. Note also that, by definition, $<$ is not reflexive. Obviously, the formula φ^{ord} needs to be satisfiable, as reported in the following lemma.

▶ **Lemma 3.5.** *The* SL *sentence* φ^{ord} *is satisfiable.*

Next two lemmas report two important properties of the formula φ^{ord}, for the negative statements we want to show. Namely, they state that, in order to be satisfied, φ^{ord} must require the existence of strict partial order relations on strategies and actions that do not admit any maximal element. From this, as stated in Theorem 3.8, we directly derive that φ^{ord} needs an infinite chain of actions to be satisfied (i.e., it cannot have a bounded model).

▶ **Lemma 3.6.** *Let* $\mathcal{G}$ *be a model of* φ^{ord} *and* $r^{<} \subseteq \mathrm{Str} \times \mathrm{Str}$ *be a relation between strategies such that* $r^{<}(\mathsf{f}_1, \mathsf{f}_2)$ *holds iff* $\mathcal{G}, \chi, s_0 \models x_1 < x_2$*, where* $\chi(x_1) = \mathsf{f}_1$ *and* $\chi(x_2) = \mathsf{f}_2$*, for all* $\chi \in \mathrm{Asg}$*, with* s_0 *as the initial state of* $\mathcal{G}$*. Then* $r^{<}$ *is a* strict partial order without maximal element.

▶ **Lemma 3.7.** *Let* $\mathcal{G}$ *be a model of* φ^{ord} *and* $s^{<} \subseteq \mathrm{Ac} \times \mathrm{Ac}$ *be a relation between actions such that* $s^{<}(c_1, c_2)$ *holds iff* $r^{<}(\mathsf{f}_1, \mathsf{f}_2)$ *holds, where* $c_1 = \mathsf{f}_1(s_0)$ *and* $c_2 = \mathsf{f}_2(s_0)$*, for all* $\mathsf{f}_1, \mathsf{f}_2 \in \mathrm{Str}$*, with* s_0 *as the initial state of* $\mathcal{G}$*. Then* $s^{<}$ *is a* strict partial order without maximal element.

Observe that the relation $s^{<}$ cannot be defined on a finite set [6]. Now, we have all tools to prove that SL lacks of the finite and bounded-tree model properties, which hold in several commonly used multi-agent logics, such as ATL*.

▶ **Theorem 3.8.** *For* SL *it holds that:* (i) *it does not have the bounded-tree model property, and* (ii) *it does not have the finite model property.*

4 Model Checking

In this section, we study the model-checking problem for SL and show that it is decidable and 2ExpTime-Complete, as for ATL*. The lower bound immediately follows from ATL*, which SL properly includes. For the upper bound, we follow an *automata-theoretic approach* [13], reducing the decision problem for the logic of interest to the emptiness problem of automata. To this aim, we use *alternating parity tree automata* (APT, for short), which are *alternating tree automata* along with a *parity acceptance condition* (see [8], for a survey). APTs are a generalization of *nondeterministic parity tree automata* (NPT, for short). Intuitively, while an NPT that visits a node of the input tree sends exactly one copy of itself to each of the successors of the node, an APT can send several copies of itself to the same successor. We recall that an approach to tree automata is only possible once the logic satisfies the tree model property. In fact, this property holds for SL as we have proved in Theorem 3.3. By the size of the automaton and the complexity required for checking its emptiness, we get the desired 2ExpTime upper bound. The definition of APTs follows.

▶ **Definition 4.1.** An APT is a tuple $\mathcal{A} = \langle \Sigma, \Delta, \mathrm{Q}, \delta, q_0, \mathrm{F} \rangle$, where Σ, Δ, and Q are non-empty finite sets of *input symbols*, *directions*, and *states*, $q_0 \in \mathrm{Q}$ is an *initial state*, $\mathrm{F} = (\mathrm{F}_1, \ldots, \mathrm{F}_k) \in (2^{\mathrm{Q}})^*$ with $\mathrm{F}_1 \subseteq \ldots \subseteq \mathrm{F}_k = \mathrm{Q}$ is a *parity acceptance condition*, and $\delta : \mathrm{Q} \times \Sigma \to \mathtt{B}^+(\Delta \times \mathrm{Q})$ is an *alternating transition function* that maps each pair of states and symbols into a Boolean positive formula on the set of propositions of the form (d, q), where d is a direction and q a state.

A *run* of an APT $\mathcal{A}$ on a Σ-labeled Δ-tree $\mathcal{T} = \langle \mathrm{T}, \mathsf{v} \rangle$ is a $(\mathrm{Q} \times \mathrm{T})$-labeled $\mathbb{N}$-tree $\mathcal{R} = \langle \mathrm{Tr}, \mathsf{r} \rangle$ such that *(i)* $\mathsf{r}(\varepsilon) = (q_0, \varepsilon)$ and *(ii)* for all $y \in \mathrm{Tr}$ with $\mathsf{r}(y) = (q, x)$, there exists a set $S \subseteq \mathrm{D} \times \mathrm{Q}$ with $S \models \delta(q, \mathsf{v}(x))$ such that, for all atoms $(d, q') \in S$, there is an index $i \in \mathbb{N}$ for which it holds that $\mathsf{r}(y \cdot i) = (q', x \cdot d)$. The run $\mathcal{R}$ is said to be *accepting* iff, for every path π, the least index $1 \leq i \leq k$ such that at least one state of F_i occurs infinitely often in π is even. The number k of sets in F is called the *index* of the automaton. A tree $\mathcal{T}$ is *accepted* by $\mathcal{A}$ if there is an accepting run of $\mathcal{A}$ on it. By $\mathrm{L}(\mathcal{A})$ we denote the language accepted by the automaton $\mathcal{A}$, i.e., the set of all trees that $\mathcal{A}$ accepts. $\mathcal{A}$ is said *empty* if $\mathrm{L}(\mathcal{A}) = \emptyset$. The *emptiness problem* for $\mathcal{A}$ is to decide whether $\mathrm{L}(\mathcal{A}) = \emptyset$.

We now proceed with the model-checking algorithm for SL. As for ATL*, we use a bottom-up model-checking algorithm, in which we start with the innermost sub-sentences and terminate with the sentence under checking. At each step, we label each state of the model with all the sub-sentences that are satisfied on it. The procedure we propose here extends that used for ATL* in [1] by means of a richer structure of the automata involved in.

First, we introduce some extra notation. A *principal sentence* φ is a sentence of the form $\mathsf{Qn}_1 x_1 \cdots \mathsf{Qn}_k x_k\, \psi_\varphi$, where $\mathsf{Qn}_i x_i \in \{\langle\!\langle x_i \rangle\!\rangle, [\![x_i]\!]\}$ and the *matrix* ψ_φ is an agent-closed formula, with $\mathsf{free}(\psi_\varphi) = \{x_1, \ldots, x_k\}$, such that it does not contain any quantification. For the sake of space and clarity of exposition, we only discuss the model checking of principal formulas. By a slight variation of both the notion of principal formulas and our procedure, we can also address the full SL. We also need the notion of atom. An *atom* ψ is an agent-closed formula of the form $(\alpha_1, y_1) \cdots (\alpha_n, y_n)\psi'$, where $\mathrm{Ag} = \{\alpha_1, \ldots, \alpha_n\}$, $y_1, \ldots, y_n$ are possible equal variables and either *(i)* ψ' does not contain any quantification and binding, i.e., it is an LTL formula, or *(ii)* the derived formula $\widehat{\psi}'$ does not contain any quantification and binding at all, where $\widehat{\psi}'$ is obtained by ψ' substituting its sub-atoms with fresh atomic propositions. W.l.o.g., we assume that each principal sentence has a matrix that is a Boolean combination of atoms. $\mathrm{Atm}(\varphi)$ denotes the set of all sub-formulas of φ that are atoms.

The core idea behind our model-checking procedure is the following. Let $\mathcal{G} = \langle \mathrm{AP}, \mathrm{Ag}, \mathrm{Ac}, \mathrm{St}, \lambda, \tau, s_0 \rangle$ be a CGS and φ be an SL principal sentence over the set $\mathrm{Ag} = \{\alpha_1, \ldots, \alpha_n\}$ of n different agents, for which we want to check if $\mathcal{G} \models \varphi$ holds or not. We first build an NPT $\mathcal{D}_\mathcal{G}$ recognizing the unwinding $\mathcal{G}^\mathcal{U}$ of $\mathcal{G}$. Then, we build an APT $\mathcal{A}'_{\mathcal{G},\varphi}$ accepting all prunings of $\mathcal{G}^\mathcal{U}$ that are coherent with the strategy quantification of φ. Such prunings are done by properly labeling its paths with elements from the set $\mathrm{Z} \triangleq \mathrm{Atm}(\varphi) \times \{start, pass\}$ of atoms associated with a flag in $\{start, pass\}$, in a way similar as it has been done for ATL* satisfiability in [20]. The *start* and *pass* flags are used to indicate whether a path guessed to satisfy at a specific state an atom $\psi \in \mathrm{Atm}(\varphi)$, starts or passes through that state, respectively. Namely, the unlabeled paths are the pruned ones that are not needed in order to satisfy the formula. Hence, $\mathcal{A}'_{\mathcal{G},\varphi}$ accepts $\mathcal{G}^\mathcal{U}$ with this additional labeling. The automata $\mathcal{D}_\mathcal{G}$ and $\mathcal{A}'_{\mathcal{G},\varphi}$ have index 2 and a number of states polynomial in the size of $\mathcal{G}$ and φ, respectively. With more details, they are both safety automata[1]. Finally, we build an APT $\mathcal{A}''_\varphi$ that checks that all paths of a pruned model accepted by $\mathcal{A}'_{\mathcal{G},\varphi}$, i.e., all labeled paths, satisfy the atoms of φ. The automaton $\mathcal{A}''_\varphi$ has index 2 and a number of states exponential in φ.

Now, recall that APTs are linearly closed under intersection. More precisely, two APTs having n_1 and n_2 states and k_1 and k_2 as indexes, respectively, can be intersected in an APT with $n_1 + n_2$ states and index $\max\{k_1, k_2\}$ [15]. So, we can build an APT $\mathcal{A}_{\mathcal{G},\varphi}$ such that $\mathrm{L}(\mathcal{A}_{\mathcal{G},\varphi}) = \mathrm{L}(\mathcal{A}'_{\mathcal{G},\varphi}) \cap \mathrm{L}(\mathcal{A}''_\varphi)$, having in particular index 2. Also, by [15], we can translate an APT with n states and index k in an equivalent NPT having $n^{\mathtt{O}(n)}$ states and index $\mathtt{O}(n)$. Hence, we can transform $\mathcal{A}_{\mathcal{G},\varphi}$ in an NPT $\mathcal{N}_{\mathcal{G},\varphi}$ with a number of states double exponential in φ and an index exponential in φ. It is well known that an NPT having n states and index k and a safety automaton with m states can be intersected in an NPT with $n \cdot m$ states and index k. Hence, by intersecting $\mathcal{D}_\mathcal{G}$ with $\mathcal{N}_{\mathcal{G},\varphi}$, we get an NPT $\mathcal{N}'_{\mathcal{G},\varphi}$ such that $\mathrm{L}(\mathcal{N}'_{\mathcal{G},\varphi}) = \mathrm{L}(\mathcal{D}_\mathcal{G}) \cap \mathrm{L}(\mathcal{N}_{\mathcal{G},\varphi})$. At this point, it is possible to prove that $\mathcal{G} \models \varphi$ iff $\mathrm{L}(\mathcal{N}'_{\mathcal{G},\varphi}) \neq \emptyset$. Observe that $\mathcal{N}'_{\mathcal{G},\varphi}$ has a number of states double exponential in φ and polynomial in $\mathcal{G}$, while it has an index exponential in φ, but independent from $\mathcal{G}$. Moreover, the automata run over the alphabet $\Sigma = \{\sigma \subseteq \mathrm{AP} \cup \mathrm{St} \cup \mathrm{Z} \mid |\sigma \cap \mathrm{St}| = 1\}$, where $|\mathrm{Z}| = \mathtt{O}(|\mathcal{G}| \times 2^{|\varphi|})$. Since the emptiness of an NPT with n states, index k, and alphabet size h can be checked in time $\mathtt{O}(h \cdot n^k)$ [12], we get that to check whether $\mathcal{G} \models \varphi$ can be done in time double exponential in φ and polynomial in $\mathcal{G}$. More precisely, the algorithm runs in $|\mathcal{G}|^{2^{\mathtt{O}(|\varphi|)}}$. The details of the automata construction follow.

[1] A safety condition is the special parity condition $(\emptyset, \mathrm{Q})$ of index 2.

The NPT $\mathcal{D}_\mathcal{G} = \langle \Sigma, \mathrm{St}, \mathrm{St}, \delta, s_0, (\emptyset, \mathrm{St}) \rangle$ has the set of directions and states formed by the states of $\mathcal{G}$ that are used to build its unwinding. Moreover, the transition function is defined as follows. At the state $s \in \mathrm{St}$, the automaton first checks that the labeling of the node of the input tree corresponds to the union of $\{s\}$ and its labeling $\lambda(s)$ in $\mathcal{G}$. Then, it sends all successors of s in the relative directions. Formally, $\delta(s,\sigma)$ is set to $\mathfrak{f}$ (false) if $\lambda(s) \cup \{s\} \neq \sigma \cap (\mathrm{AP} \cup \mathrm{St})$ and to $\bigwedge_{s' \in \{\tau(s,\mathsf{d}) \mid \mathsf{d} \in \mathrm{Dc}\}} (s', s')$ otherwise. Note that $|\mathcal{D}_\mathcal{G}| = \mathtt{O}(|\mathcal{G}|)$.

The APT $\mathcal{A}'_{\mathcal{G},\varphi} = \langle \Sigma, \mathrm{St}, \{q_0\} \cup \mathrm{Atm}(\varphi), \delta, q_0, (\emptyset, \{q_0\} \cup \mathrm{Atm}(\varphi)) \rangle$ has the set of states formed by a distinguished state q_0, which is also initial, and from the atoms in $\mathrm{Atm}(\varphi)$ that are used to verify the correctness of the additional labeling Z. Moreover, the transition function is defined as follows. $\delta(\psi,\sigma)$ is equal to $\mathfrak{t}$ (true) if $(\psi, \mathit{pass}) \in \sigma \cap \mathrm{Z}$ and to $\mathfrak{f}$ (false) otherwise. The automaton at state q_0 sends the same state in all the directions individuated by the quantification, together with the control state ψ. It is important to note that the quantification here is reproduced by conjunctions and disjunctions on all possible actions of $\mathcal{G}$. Formally, $\delta(q_0,\sigma)$ is set to $\mathsf{Op}_{1\,c_1 \in \mathrm{Ac}} \cdots \mathsf{Op}_{k\,c_k \in \mathrm{Ac}} \bigwedge_{(\psi,\star) \in \sigma \cap \mathrm{Z}} (\tau(s,\mathsf{d}), q_0) \wedge (\tau(s,\mathsf{d}), \psi)$, where $\mathsf{Op}_{i\,c_i \in \mathrm{Ac}}$ is a disjunction if $\mathsf{Qn}_i x_i = \langle\!\langle x_i \rangle\!\rangle$ and a conjunction if $\mathsf{Qn}_i x_i = [\![x_i]\!]$, $\{s\} = \sigma \cap \mathrm{St}$, and $\mathsf{d}(\alpha_i) = c_j$ iff in the atom ψ the binding (α_i, x_j) appears. Note that $|\mathcal{A}'_{\mathcal{G},\varphi}| = \mathtt{O}(|\varphi|)$.

Finally, we build the APT $\mathcal{A}''_\varphi$. Let $\widehat{\psi}$ be the LTL formula obtained by replacing in $\psi \in \mathrm{Atm}(\varphi)$ all the occurrences of each other atom $\psi' \in \mathrm{Atm}(\psi)$ with the fresh atomic proposition (ψ', start). By using a slight variation of the procedure developed in [21], we can translate $\widehat{\psi}$ into a universal co-Büchi word automaton[2] $\mathcal{U}_\psi = \langle \Sigma, \mathrm{Q}_\psi, \delta_\psi, \mathrm{Q}_{0\psi}, \mathrm{F}_\psi \rangle$, with a number of states at most exponential in $|\psi|$, accepting the infinite words on Σ that are models of $\widehat{\psi}$. At this point, we can construct the automaton $\mathcal{A}''_\varphi$ that recognizes the trees whose paths, labeled with the flags $(\psi, \star)$, for $\star \in \{\mathit{start}, \mathit{pass}\}$, and starting with the label (ψ, start), satisfy the LTL formula $\widehat{\psi}$, for all $\psi \in \mathrm{Atm}(\varphi)$.

Formally, $\mathcal{A}''_\varphi = \langle \Sigma, \mathrm{St}, \{q_0, q_c\} \cup \mathrm{Q}, \delta, q_0, (\mathrm{F}, \{q_0, q_c\} \cup \mathrm{Q}) \rangle$ is built as follows. $\mathrm{Q} = \bigcup_{\psi \in \mathrm{Atm}(\varphi)} \{\psi\} \times \mathrm{Q}_\psi$ and $\mathrm{F} = \bigcup_{\psi \in \mathrm{Atm}(\varphi)} \{\psi\} \times \mathrm{F}_\psi$ are, respectively, the disjoint union of the set of states and final states of the word automata $\mathcal{U}_\psi$, for every atom $\psi \in \mathrm{Atm}(\varphi)$. q_0 is the *initial state* used to verify that the formula ψ_φ (the matrix of φ) holds at the root of the tree in input, by checking whether the labeling of the root contains all the propositions required by ψ_φ to hold. If the checking succeeds, q_0 behaves as the state q_c. Formally, let ψ_φ be considered as a boolean formula on the set of atoms $\mathrm{Atm}(\varphi)$ in which we assume $\psi = (\psi, \mathit{start})$, for all $\psi \in \mathrm{Atm}(\varphi)$. Then, $\delta(q_0,\sigma)$ is set to $\delta(q_c,\sigma)$, if $\sigma \cap \mathrm{Z} \models \psi_\varphi$ and to $\mathfrak{f}$ (false), otherwise. q_c is the *checking state* used to start the verification of the atoms ψ in every node of the input tree that contains the flag (ψ, start), which indicates the existence of a path starting in that node that satisfies ψ. To do this, q_c sends in all the directions *(i)* a copy of the state itself, to continue the control on the remaining part of the tree, and *(ii)* the states derived by all initial states of the automata $\mathcal{U}_\psi$, for all the atoms ψ for which a flag (ψ, start) appears in the labeling σ. Formally, $\delta(q_c,\sigma)$ is $\bigwedge_{s \in \mathrm{St}} (s, q_c) \wedge \bigwedge_{(\psi,\mathit{start}) \in \sigma \cap \mathrm{Z}} \bigwedge_{q \in \mathrm{Q}_{0\psi}} \bigwedge_{q' \in \delta_\psi(q, \sigma \cap \mathrm{AP})} (s, (\psi, q'))$. The states of the form (ψ, q) are used to run $\mathcal{U}_\psi$ on all paths labeled by the related flags (ψ, pass). Formally, $\delta((\psi, q), \sigma)$ is set to $\mathfrak{t}$ (true) if $(\psi, \mathit{pass}) \notin \sigma \cap \mathrm{Z}$ and to $\bigwedge_{s \in \mathrm{St}} \bigwedge_{q' \in \delta_\psi(q, \sigma \cap \mathrm{AP})} (s, (\psi, q'))$ otherwise. Note that $|\mathcal{A}''_\varphi| = \mathtt{O}(2^{|\varphi|})$.

By a simple calculation, it follows that the overall procedure results in an algorithm that is in PTIME w.r.t the size of $\mathcal{G}$ and in 2EXPTIME w.r.t. the size of φ. Hence, by getting lower bounds from ATL*, the following result holds.

▶ **Theorem 4.2.** *The model-checking problem for* SL *is* PTIME *w.r.t. the size of the model and* 2EXPTIME-COMPLETE *w.r.t the size of the specification.*

We conclude this section by pointing out that the model checking procedure described above for SL is completely different from that one used in [3] for $\mathrm{SL}_{\mathrm{CHP}}$. Indeed in [3], the authors use

[2] Word automata can be seen as tree automata in which the tree has just one path. A universal word automaton is a particular case of alternating automata in which there is no nondeterminism. A co-Büchi acceptance condition $\mathrm{F} \subseteq \mathrm{Q}$ is the special parity condition (F, Q) of index 2.

a top-down approach and, most important, for every quantification in the formula, they make a projection of the automaton they build at each stage (one for each quantification). Since at each projection they have an exponential blow-up, at the end their procedure results in a non-elementary one, both in the size of the system and the formula. Our iterative approach, instead, does not make use of any projection, since we reduce strategy quantifications to action quantifications, which, as we have stated, can be handled locally on each state of the model.

5 Satisfiability

In this section, we show the undecidability of the satisfiability problem for SL through a reduction of the *recurrent domino problem*. In particular, as we discuss later, the reduction also holds for SL_{CHP} under the concurrent game semantics.

The *domino problem*, proposed for the first time by Wang [22], consists of placing a given number of tile types on an infinite grid, satisfying a predetermined set of constraints on adjacent tiles. Its standard version asks for a compatible tiling of the whole plane $\mathbb{N}\times\mathbb{N}$. The *recurrent domino problem* further requires the existence of a distinguished tile type that occurs infinitely often in the first row of the grid. This problem was proved to be highly undecidable by Harel, and in particular, Σ^1_1-COMPLETE [9]. The formal definition follows.

▶ **Definition 5.1** (Recurrent Domino System)**.** An $\mathbb{N}\times\mathbb{N}$ *recurrent domino system* $\mathcal{D}=\langle \mathrm{D}, H, V, t^*\rangle$ consists of a finite non-empty set D of *domino types*, two *horizontal* and *vertical matching relations* $H, V \subseteq \mathrm{D}\times\mathrm{D}$, and a distinguished tile type $t^*\in\mathrm{D}$. The recurrent domino problem asks for an *admissible tiling* of $\mathbb{N}\times\mathbb{N}$, which is a *solution mapping* $\partial:\mathbb{N}\times\mathbb{N}\to\mathrm{D}$ such that, for all $x,y\in\mathbb{N}$, it holds that *(i)* $(\partial(x,y),\partial(x+1,y))\in H$, *(ii)* $(\partial(x,y),\partial(x,y+1))\in V$, and *(iii)* $|\{x \mid \partial(x,0)=t^*\}|=\infty$.

By showing a reduction from the recurrent domino problem, we prove that the satisfiability problem for SL is Σ^1_1-HARD, which implies that it is even not computably enumerable. We achieve this reduction by showing that a given recurrent tiling system $\mathcal{D}=\langle \mathrm{D}, H, V, t^*\rangle$ can be *"embedded"* into a model of a particular sentence $\varphi^{dom} \triangleq \varphi^{grd}\wedge\varphi^{til}\wedge\varphi^{rec}$ over $\mathrm{AP}=\{p\}\cup\mathrm{D}$ and $\mathrm{Ag}=\{\alpha,\beta\}$, where $p\notin\mathrm{D}$, in such a way that φ^{dom} is satisfiable iff $\mathcal{D}$ allows an admissible tiling. For the sake of clarity, we split the reduction into three tasks where we explicit the sentences φ^{grd}, φ^{til}, and φ^{rec}.

Grid specification. Consider the sentence $\varphi^{grd}\triangleq\bigwedge_{a\in\mathrm{Ag}}\varphi^{ord}_a$, where $\varphi^{ord}_a=\varphi^{unb}_a\wedge\varphi^{trn}_a$ are *order sentences* and φ^{exs}_a and φ^{trn}_a are the *strategy unboundedness* and *strategy transitivity* requirements for agents α and β defined, similarly in Definition 3.4, as follows:

1. $\varphi^{unb}_a\triangleq[\![z_1]\!]\langle\!\langle z_2\rangle\!\rangle\, z_1<_a z_2$,
2. $\varphi^{trn}_a\triangleq[\![z_1]\!][\![z_2]\!][\![z_3]\!]\,(z_1<_a z_2\wedge z_2<_a z_3)\to z_1<_a z_3$,

where $x_1<_\alpha x_2\triangleq\langle\!\langle y\rangle\!\rangle\,(\beta,y)((\alpha,x_1)(\mathsf{X}\,p)\wedge(\alpha,x_2)(\mathsf{X}\neg p))$ and $y_1<_\beta y_2\triangleq\langle\!\langle x\rangle\!\rangle\,(\alpha,x)((\beta,y_1)(\mathsf{X}\neg p)\wedge(\beta,y_2)(\mathsf{X}\,p))$ are the two *partial order* formulas on strategies of α and β, respectively. Intuitively, $<_\alpha$ and $<_\beta$ correspond to the horizontal and vertical ordering of the positions in the grid, respectively.

It easy to see that φ^{grd} is satisfiable, as it follows from the same argument used in Lemma 3.5.

▶ **Lemma 5.2.** *The* SL *sentence* φ^{grd} *is satisfiable on a* CGS *with a countable number of actions.*

Consider now a model $\mathcal{G}=\langle \mathrm{AP},\mathrm{Ag},\mathrm{Ac},\mathrm{St},\lambda,\tau,s_0\rangle$ of φ^{grd} and the relations $r^<_a\subseteq\mathrm{Str}\times\mathrm{Str}$ between strategies, for all agents $a\in\mathrm{Ag}$, defined as follows: $r^<_a(\mathsf{f}_1,\mathsf{f}_2)$ holds iff $\mathcal{G},\chi,s_0\models z_1<_a z_2$, where $\chi(z_1)=\mathsf{f}_1$ and $\chi(z_2)=\mathsf{f}_2$. By Lemma 3.6, the relations $r^<_a$ are *strict partial orders without maximal element* on Str. To apply the desired reduction, we need to transform $r^<_a$ into total orders over strategies. Let $r^{\equiv}_a\subseteq\mathrm{Str}\times\mathrm{Str}$, with $a\in\mathrm{Ag}$, be the two relations between strategies such that $r^{\equiv}_a(\mathsf{f}_1,\mathsf{f}_2)$ holds iff neither $r^<_a(\mathsf{f}_1,\mathsf{f}_2)$ nor $r^<_a(\mathsf{f}_2,\mathsf{f}_1)$ holds. It is possible to show that $r^{\equiv}_a$ are *equivalence relations*. Now, let $\mathrm{Str}^{\equiv}_a=\mathrm{Str}/r^{\equiv}_a$ be the quotient sets of Str w.r.t. $r^{\equiv}_a$, i.e., the sets of the related equivalence

classes over strategies. Also, let $s_a^{<} \subseteq \mathrm{Str}_a^{\equiv} \times \mathrm{Str}_a^{\equiv}$, with $a \in \mathrm{Ag}$, be the two relations between classes of strategies such that $s_a^{<}(F_1, F_2)$ holds iff, for all $\mathsf{f}_1 \in F_1$ and $\mathsf{f}_2 \in F_2$, it holds that $r_a^{<}(\mathsf{f}_1, \mathsf{f}_2)$. Then, it is easy to prove that $s_a^{<}$ are *strict total orders with minimal element but no maximal element*. By a classical result on first order logic model theory [6], $s_a^{<}$ cannot be defined on a finite set. Hence, $|\mathrm{Str}_a^{\equiv}| = \infty$, for all $a \in \mathrm{Ag}$. Now, let $s_a^{\prec}$ be the *successor* relations on $\mathrm{Str}_a^{\equiv}$ compatible with the strict total orders $s_a^{<}$, i.e., such that $s_a^{\prec}(F_1, F_2)$ holds iff *(i)* $s_a^{<}(F_1, F_2)$ holds and *(ii)* there is no $F_3 \in \mathrm{Str}_a^{\equiv}$ for which both $s_a^{<}(F_1, F_3)$ and $s_a^{<}(F_3, F_2)$ hold, for all $F_1, F_2 \in \mathrm{Str}_a^{\equiv}$. Then, we can write the two sets of classes $\mathrm{Str}_\alpha^{\equiv}$ and $\mathrm{Str}_\beta^{\equiv}$ as the infinite ordered lists $\{F_0^\alpha, F_1^\alpha, \ldots\}$ and $\{F_0^\beta, F_1^\beta, \ldots\}$, respectively, such that $s_a^{\prec}(F_i^a, F_{i+1}^a)$, for all $a \in \mathrm{Ag}$ and $i \in \mathbb{N}$. Note that F_0^a are the classes of minimal strategies w.r.t the relations $s_a^{<}$.

Now, we have all the machinery to build an embedding of the plane $\mathbb{N} \times \mathbb{N}$ into the model $\mathcal{G}$ of φ^{grd}. In particular, we are able to construct a *bijective map* $\aleph : \mathbb{N} \times \mathbb{N} \to \mathrm{Str}_\alpha^{\equiv} \times \mathrm{Str}_\beta^{\equiv}$ such that $\aleph(i, j) = (F_i^\alpha, F_j^\beta)$, for all $i, j \in \mathbb{N}$.

Compatible tiling. Given the grid structure built on the model $\mathcal{G}$ of φ^{grd} through the bijective map $\aleph$, we can express that a tiling of the grid is admissible by making use of the formulas $z_1 \prec_a z_2 \triangleq z_1 <_a z_2 \wedge \neg\langle\!\langle z_3 \rangle\!\rangle\, z_1 <_a z_3 \wedge z_3 <_a z_2$ corresponding to the successor relations $s_a^{\prec}$, for all $a \in \mathrm{Ag}$. Indeed, it is not hard to see that $\mathcal{G}, \chi, \varepsilon \models z_1 \prec_a z_2$ holds iff $\chi(z_1) \in F_i^a$ and $\chi(z_2) \in F_{i+1}^a$, for all $i \in \mathbb{N}$. The idea here is to associate to each domino type $t \in \mathrm{D}$ a corresponding atomic proposition $t \in \mathrm{AP}$ and to express the horizontal and vertical matching conditions via suitable object labeling. In particular, we can express that the tiling is locally compatible, that the horizontal neighborhood of a tile satisfies the H requirement, and that also its vertical neighborhood satisfies the V requirement, all through the following three agent-closed formulas, respectively:

1. $\varphi^{t,loc}(x,y) \triangleq (\alpha,x)(\beta,y)(\mathsf{X}\,(t \wedge \bigwedge_{t' \in \mathrm{D}}^{t' \neq t} \neg t'))$;
2. $\varphi^{t,hor}(x,y) \triangleq \bigvee_{(t,t') \in H} [\![x']\!]\, x \prec_\alpha x' \to (\alpha,x')(\beta,y)(\mathsf{X}t')$;
3. $\varphi^{t,ver}(x,y) \triangleq \bigvee_{(t,t') \in V} [\![y']\!]\, y \prec_\beta y' \to (\alpha,x)(\beta,y')(\mathsf{X}t')$.

Informally, we have the following: $\varphi^{t,loc}(x,y)$ asserts that t is the only domino type labeling the successors of the root of the model $\mathcal{G}$ that can be reached using the strategies related to the variables x and y; $\varphi^{t,hor}(x,y)$ asserts that the tile t' labeling the successors of the root reachable through the strategies x' and y is compatible with t w.r.t. the horizontal requirement H, for all strategies x' that immediately follow that related to x w.r.t. the order $r_\alpha^{<}$; $\varphi^{t,ver}(x,y)$ asserts that the tile t' labeling the successors of the root reachable through the strategies x and y' is compatible with t w.r.t. the vertical requirement V, for all strategies y' that immediately follow that related to y w.r.t. the order $r_\beta^{<}$.

Finally, to express that the whole grid has an admissible tiling, we use the sentence $\varphi^{til} \triangleq [\![x]\!][\![y]\!] \bigvee_{t \in \mathrm{D}} \varphi^{t,loc}(x,y) \wedge \varphi^{t,hor}(x,y) \wedge \varphi^{t,ver}(x,y)$ that asserts, for every point individuated by the strategies x and y, the existence of a domino type t satisfying the three conditions mentioned above.

Recurrent tile. As last task, we impose that the grid embedded into $\mathcal{G}$ has the distinguished domino type t^* occurring infinitely often in its first row. To do this, we first use two formulas that determine if a row or a column is the first one or not w.r.t. the orders $s_\alpha^{<}$ and $s_\beta^{<}$, respectively. Formally, we use $0_a(z) \triangleq \neg\langle\!\langle z' \rangle\!\rangle\, z' <_a z$, for $a \in \mathrm{Ag}$. One can prove that $\mathcal{G}, \chi, \varepsilon \models 0_\alpha(z)$ iff $\chi(z) \in F_0^a$.

Now, the infinite occurrence requirement on t^* can be expressed with the following sentence: $\varphi^{rec} \triangleq [\![x]\!][\![y]\!]\,(0_\beta(y) \wedge (0_\alpha(x) \vee (\alpha,x)(\beta,y)(\mathsf{X}t^*))) \to \langle\!\langle x' \rangle\!\rangle\, x <_\alpha x' \wedge (\alpha,x')(\beta,y)(\mathsf{X}t^*)$. Informally, φ^{rec} asserts that, when we are on the first row individuated by the variable y and at a column individuated by x such that it is the first column or the node of the "intersection" between x and y is labeled by t^*, we have that there exists a greater column individuated by x' such that its "intersection" with y is labeled by t^* as well.

Construction correctness. Now we have all the tools to formally prove the correctness of the undecidability reduction, by showing the equivalence between finding the solution of the recurrent tiling problem and the satisfiability of the sentence φ^{dom}. In particular, one can note that in the reduction we propose, only the $\mathrm{SL}_{\mathrm{CHP}}$ fragment of SL is involved. Thus, we prove that $\mathrm{SL}_{\mathrm{CHP}}$ under the concurrent semantics is highly undecidable, while it remains an open question whether this problem is undecidable or not in the turned-based framework.

▶ **Theorem 5.3.** *The satisfiability problem for* $\mathrm{SL}_{\mathrm{CHP}}$ *under the concurrent semantics is highly undecidable. In particular, it is* Σ^1_1-HARD.

References

1 R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-Time Temporal Logic. *JACM*, 49(5):672–713, 2002.

2 T. Brihaye, A.D.C. Lopes, F. Laroussinie, and N. Markey. ATL with Strategy Contexts and Bounded Memory. In *LFCS'09*, LNCS 5407, pages 92–106. Springer, 2009.

3 K. Chatterjee, T.A. Henzinger, and N. Piterman. Strategy Logic. In *CONCUR'07*, LNCS 4703, pages 59–73. Springer, 2007.

4 E.M. Clarke and E.A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *LP'81*, LNCS 131, pages 52–71. Springer, 1981.

5 E.M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking.* MIT Press, 2002.

6 H.D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

7 E.A. Emerson and J.Y. Halpern. "Sometimes" and "Not Never" Revisited: On Branching Versus Linear Time. *JACM*, 33(1):151–178, 1986.

8 E. Grädel, W. Thomas, and T. Wilke. *Automata, Logics, and Infinite Games: A Guide to Current Research.* LNCS 2500. Springer, 2002.

9 D. Harel. A Simple Highly Undecidable Domino Problem. In *LCC'84*, 1984.

10 W. Jamroga and W. van der Hoek. Agents that Know How to Play. *FI*, 63(2-3):185–219, 2004.

11 D. Kozen. Results on the Propositional mu-Calculus. *TCS*, 27:333–354, 1983.

12 O. Kupferman and M.Y. Vardi. Weak Alternating Automata and Tree Automata Emptiness. In *STOC'98*, pages 224–233, 1998.

13 O. Kupferman, M.Y. Vardi, and P. Wolper. An Automata Theoretic Approach to Branching-Time Model Checking. *JACM*, 47(2):312–360, 2000.

14 O. Kupferman, M.Y. Vardi, and P. Wolper. Module Checking. *IC*, 164(2):322–344, 2001.

15 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. *TCS*, 141:69–107, 1995.

16 M. Pauly. A Modal Logic for Coalitional Power in Games. *JLC*, 12(1):149–166, 2002.

17 S. Pinchinat. A Generic Constructive Solution for Concurrent Games with Expressive Constraints on Strategies. In *ATVA'07*, LNCS 4762, pages 253–267. Springer, 2007.

18 A. Pnueli. The Temporal Logic of Programs. In *FOCS'77*, pages 46–57, 1977.

19 J.P. Queille and J. Sifakis. Specification and Verification of Concurrent Programs in Cesar. In *SP'81*, LNCS 137, pages 337–351. Springer, 1981.

20 S. Schewe. ATL* Satisfiability is 2ExpTime-Complete. In *ICALP'08*, LNCS 5126, pages 373–385. Springer, 2008.

21 M.Y. Vardi and P. Wolper. An Automata-Theoretic Approach to Automatic Program Verification. In *LICS'86*, pages 332–344. IEEE Computer Society, 1986.

22 H. Wang. Proving Theorems by Pattern Recognition II. *BSTJ*, 40:1–41, 1961.

New Results on Quantum Property Testing*

Sourav Chakraborty[1], Eldar Fischer[2], Arie Matsliah[3], and Ronald de Wolf[3]

1 **Chennai Mathematical Institute, Chennai, India.**
`sourav@cmi.ac.in`
2 **Computer Science Faculty, Israel Institute of Technology (Technion).**
`eldar@cs.technion.ac.il`
3 **Centrum Wiskunde & Informatica, Amsterdam.**
`{ariem,rdewolf}@cwi.nl`

Abstract

We present several new examples of speed-ups obtainable by quantum algorithms in the context of property testing.

First, motivated by sampling algorithms, we consider probability distributions given in the form of an oracle $f : [n] \to [m]$. Here the probability $\mathcal{P}_f(j)$ of an outcome $j \in [m]$ is the fraction of its domain that f maps to j. We give quantum algorithms for testing whether two such distributions are identical or ϵ-far in L_1-norm. Recently, Bravyi, Hassidim, and Harrow [11] showed that if $\mathcal{P}_f$ and $\mathcal{P}_g$ are both unknown (i.e., given by oracles f and g), then this testing can be done in roughly $\sqrt{m}$ quantum queries to the functions. We consider the case where the second distribution is known, and show that testing can be done with roughly $m^{1/3}$ quantum queries, which we prove to be essentially optimal. In contrast, it is known that classical testing algorithms need about $m^{2/3}$ queries in the unknown-unknown case and about $\sqrt{m}$ queries in the known-unknown case. Based on this result, we also reduce the query complexity of graph isomorphism testers with quantum oracle access.

While those examples provide polynomial quantum speed-ups, our third example gives a much larger improvement (constant quantum queries vs polynomial classical queries) for the problem of testing periodicity, based on Shor's algorithm and a modification of a classical lower bound by Lachish and Newman [27]. This provides an alternative to a recent constant-vs-polynomial speed-up due to Aaronson [1].

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.145

1 Introduction

Since the early 1990s, a number of *quantum* algorithms have been discovered that have much better query complexity than their best classical counterparts [15, 31, 22, 4, 16, 5]. Around the same time, the area of *property testing* gained prominence [9, 20, 18, 29]. Here the aim is to design algorithms that can efficiently test whether a given very large piece of data satisfies some specific property, or is "far" from having that property.

Buhrman et al. [12] combined these two strands, exhibiting various testing problems where quantum testers are much more efficient than classical testers. There has been some recent subsequent work on quantum property testing, such as the work of Friedl et al. [19] on testing hidden group properties, Atici and Servedio [6] on testing juntas, Inui and Le Gall [25]

* This work was partially supported by an ERC-2007-StG grant number 202405-2 and by an ISF grant number 1101/06 and was also partially supported by a Vidi grant from the Netherlands Organization for Scientific Research (NWO), and by the European Commission under the Integrated Project Qubit Applications (QAP) funded by the IST directorate as Contract Number 015848.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 145–156

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

on testing group solvability, Childs and Liu [14] on testing bipartiteness and expansion, Aaronson [1] on "Fourier checking", and Bravyi, Hassidim, and Harrow [11] on testing distributions.

In this paper we continue this line of research, coming up with a number of new examples where quantum testers substantially improve upon their classical counterparts. It should be noted that we do not invent new quantum algorithms here—rather, we use known quantum algorithms as subroutines in otherwise *classical* testing algorithms.

1.1 Distribution Testing

How many samples are needed to determine whether two distributions are identical or have L_1-distance more than ϵ? This is a fundamental problem in statistical hypothesis testing and also arises in other subjects like property testing and machine learning.

We use the notation $[n] = \{1, 2, 3, \ldots, n\}$. For a function $f : [n] \to [m]$, we denote by $\mathcal{P}_f$ the distribution over $[m]$ in which the weight $\mathcal{P}_f(j)$ of every $j \in [m]$ is proportional to the number of elements $i \in [n]$ that are mapped to j. We use this form of representation for distributions in order to allow *queries*. Namely, we assume that the function $f : [n] \to [m]$ is accessible by an oracle of the form $|x\rangle|b\rangle \mapsto |x\rangle|b \oplus f(x)\rangle$, where x is a $\log n$-bit string, b and $f(x)$ are $\log m$-bit strings and $\oplus$ is bitwise addition modulo two. Note that a classical random sample according to a distribution $\mathcal{P}_f$ can be simply obtained by picking $i \in [n]$ uniformly at random and evaluating $f(i)$. In fact, a classical algorithm cannot make a better use of the oracle, since the actual labels of the domain $[n]$ are irrelevant.

We say that the distribution $\mathcal{P}_f$ is *known* (or *explicit*) if the function f is given explicitly, and hence all probabilities $\mathcal{P}_f(j)$ can be computed. $\mathcal{P}_f$ is *unknown* (or *black-box*) if we only have oracle access to the function f, and no additional information about f is given. Two distributions $\mathcal{P}_f, \mathcal{P}_g$ defined by functions $f, g : [n] \to [m]$ are *ϵ-far* if the L_1-distance between them is at least ϵ, i.e., $\|\mathcal{P}_f - \mathcal{P}_g\|_1 = \sum_{j=1}^{m} |\mathcal{P}_f(j) - \mathcal{P}_g(j)| \geq \epsilon$. Note that $f = g$ implies $\mathcal{P}_f = \mathcal{P}_g$ but not vice versa (for instance, permuting f leaves $\mathcal{P}_f$ invariant). Two problems of testing distributions can be formally stated as follows:

- **unknown-unknown case.** Given n, m, ϵ and oracle access to $f, g : [n] \to [m]$, how many queries to f and g are required in order to determine whether the distributions $\mathcal{P}_f$ and $\mathcal{P}_g$ are identical or ϵ-far?
- **known-unknown case.** Given n, m, ϵ, oracle access to $f : [n] \to [m]$ and a known distribution $\mathcal{P}_g$ (defined by an explicitly given function $g : [n] \to [m]$), how many queries to f are required to determine whether $\mathcal{P}_f$ and $\mathcal{P}_g$ are identical or ϵ-far?

If only *classical* queries are allowed (where querying the distribution means asking for a random sample), the answers to these problems are well known. For the unknown-unknown case Batu, Fortnow, Rubinfeld, Smith, and White [8] proved an upper bound of $\widetilde{O}(m^{2/3})$ on the query complexity, and Valiant [32] proved a matching (up to polylogarithmic factors) lower bound. For the known-unknown case, Goldreich and Ron [21] showed a lower bound of $\Omega(\sqrt{m})$ queries and Batu, Fischer, Fortnow, Rubinfeld, Smith, and White [7] proved a nearly tight upper bound of $\widetilde{O}(\sqrt{m})$ queries.[1]

[1] These classical lower bounds are stated in terms of number of samples rather than number of queries, but it is not hard to see that they hold in both models. In fact, the $\sqrt{m}$ classical query lower bound for the known-unknown case follows by the same argument as the quantum lower bound.

1.1.1 Testing with Quantum Queries

Allowing quantum queries for accessing distributions, Bravyi, Hassidim, and Harrow [11] recently showed that the L_1-distance between two unknown distributions can actually be estimated up to small error with only $O(\sqrt{m})$ queries. Their result implies an $O(\sqrt{m})$ upper bound on the quantum query complexity for the unknown-unknown testing problem defined above. In this paper we consider the known-unknown case, and prove nearly tight bounds on its quantum query complexity.

▶ **Theorem 1.** *Given n, m, ϵ, oracle access to $f : [n] \to [m]$ and a known distribution $\mathcal{P}_g$ (defined by an explicitly given function $g : [n] \to [m]$), the quantum query complexity of determining whether $\mathcal{P}_f$ and $\mathcal{P}_g$ are identical or ϵ-far is $O(\frac{m^{1/3} \log^2 m \log \log m}{\epsilon^5}) = m^{1/3} \cdot \mathrm{poly}(\frac{1}{\epsilon}, \log m)$.*

We prove Theorem 1 in two parts. First, in Section 3.1, we prove that with $O(\frac{m^{1/3}}{\epsilon^2})$ quantum queries it is possible to test whether a black-box distribution $\mathcal{P}_f$ (defined by some $f : [n] \to [m]$) is ϵ-close to uniform. We actually prove that this can be even done *tolerantly* in a sense, meaning that a distribution that is close to uniform in the L_∞ norm is accepted with high probability (see Theorem 10 for the formal statement). Then, in Section 3.2, we use the bucketing technique (see Section 2.1) to reduce the task of testing closeness to a known distribution to testing uniformity.

We stress that the main difference between the classical algorithm of [7] and ours is that in [7] they check the "uniformity" of the unknown distribution in every bucket by approximating the corresponding L_2 norms of the conditional distributions. It is not clear if one can gain anything (in the quantum case) using the same strategy, since we are not aware of any quantum procedure that can approximate the L_2 norm of a distribution with less than $\sqrt{m}$ queries. Hence, we reduce the main problem *directly* to the problem of testing uniformity. For this reduction to work, the uniformity tester has to be tolerant in the sense mentioned above (see Section 3.2 for details).

A different quantum uniformity tester was recently discovered (independently) in [11]. We note that our version has the advantages of being tolerant, which is crucial for the application above, and it has only polynomial dependence on ϵ (instead of exponential), which is essentially optimal.

1.1.2 Quantum Lower Bounds

Known quantum query lower bounds for the collision problem [2, 3, 26] imply that in both known-unknown and unknown-unknown cases roughly $m^{1/3}$ quantum queries are required. In fact, the lower bound applies even for testing uniformity (proof omitted from this extended abstract):

▶ **Theorem 2.** *Given n, m, ϵ and oracle access to $f : [n] \to [m]$, the quantum query complexity of determining whether $\mathcal{P}_f$ is uniform or ϵ-far from uniform is $\Omega(m^{1/3})$.*

The main remaining open problem is to tighten the bounds on the quantum query complexity for the unknown-unknown case. It would be very interesting if this case could also be tested using roughly $m^{1/3}$ quantum queries. In fact the easiest way to do this (just reconstructing both unknown distributions up to small error) will not work—it requires $\Omega(m/\log m)$ quantum queries.

1.2 Graph Isomorphism Testing

Fischer and Matsliah [17] studied the problem of testing graph isomorphism in the dense-graph model, where the graphs are represented by their adjacency matrices, and querying the graph corresponds to reading a single entry from its adjacency matrix. The goal in isomorphism testing is to determine, with high probability, whether two graphs G and H are isomorphic or ϵ-far from being isomorphic, making as few queries as possible. (The graphs are ϵ-far from being isomorphic if at least an ϵ-fraction of the entries in their adjacency matrices need to be modified in order to make them isomorphic.)

In [17] two models were considered:

- **unknown-unknown case.** Both G and H are unknown, and they can only be accessed by querying their adjacency matrices.
- **known-unknown case.** The graph H is known (given in advance to the tester), and the graph G is unknown (can only be accessed by querying its adjacency matrix).

As usual, in both models the query complexity is the worst-case number of queries needed to test whether the graphs are isomorphic. [17] give nearly tight bounds of $\widetilde{\Theta}(\sqrt{|V|})$ on the (classical) query complexity in the known-unknown model. For the unknown-unknown model they prove an upper bound of $\widetilde{O}(|V|^{5/4})$ and a lower bound of $\Omega(|V|)$ on the query complexity.

Allowing quantum queries[2], we can use our aforementioned results to prove the following query-complexity bounds for testing graph isomorphism (proof omitted from this extended abstract):

▶ **Theorem 3.** *The quantum query complexity of testing graph isomorphism in the known-unknown case is* $\widetilde{\Theta}(|V|^{1/3})$*, and in the unknown-unknown case it is between* $\Omega(|V|^{1/3})$ *and* $\widetilde{\Theta}(|V|^{7/6})$.

1.3 Periodicity Testing

The quantum testers mentioned above obtain polynomial speed-ups over their classical counterparts, and that is the best one can hope to obtain for these problems. The paper by Buhrman et al. [12], which first studied quantum property testing, actually provides two super-polynomial separations between quantum and classical testers: a constant-vs-$\log n$ separation based on the Bernstein-Vazirani algorithm, and a (roughly) $\log n$-vs-$\sqrt{n}$ separation based on Simon's algorithm. They posed as an open problem whether there exists a constant-vs-n separation. Recently, in an attempt to construct oracles to separate BQP from the Polynomial Hierarchy, Aaronson [1] analyzed the problem of "Fourier checking": roughly, the input consists of two m-bit Boolean functions f and g, such that g is either strongly or weakly correlated with the Fourier transform of f (i.e., $g(x) = \text{sign}(\hat{f}(x))$ either for most x or for roughly half of the x). He proved that quantum algorithms can decide this with $O(1)$ queries while classical algorithms need $\Omega(2^{m/4})$ queries. Viewed as a testing problem on an input of length $n = 2 \cdot 2^m$ bits, this is the first constant-vs-polynomial separation between quantum and classical testers.

In Section 4 we obtain another separation that is (roughly) constant-vs-$n^{1/4}$. Our testing problem is reverse-engineered from the periodicity problem solved by Shor's famous factoring algorithm [30]. Suppose we are given a function $f : [n] \to [m]$, which we can query in the

[2] A quantum query to the adjacency matrix of a graph G can be of the form $|i,j\rangle|b\rangle \mapsto |i,j\rangle|b \oplus G(i,j)\rangle$, where $G(i,j)$ is the (i,j)-th entry of the adjacency matrix of G and $\oplus$ is addition modulo two.

usual way. We call f *1-1-p-periodic* if the function is injective on $[p]$ and repeats afterwards. Equivalently:

$f(i) = f(j)$ iff $i = j \bmod p$.

Note that we need $m \geq p$ to make this possible. In fact, for simplicity we will assume $m \geq n$. Let $\mathcal{P}_p$ be the set of functions $f : [n] \to [m]$ that are 1-1-p-periodic, and $\mathcal{P}_{q,r} = \cup_{p=q}^{r} \mathcal{P}_p$. The 1-1-PERIODICITY TESTING problem, with parameters $q \leq r$ and small fixed constant ϵ, is as follows:

given an f which is either in $\mathcal{P}_{q,r}$ or ϵ-far from $\mathcal{P}_{q,r}$, find out which is the case.

Note that *for a given* p it is easy to test whether f is p-periodic or ϵ-far from it: choose an $i \in [p]$ uniformly at random, and test whether $f(i) = f(i + kp)$ for a random positive integer k. If f is p-periodic then these values will be the same, but if f is ϵ-far from p-periodic then we will detect this with constant probability. However, $r - q + 1$ different values of p are possible in $\mathcal{P}_{q,r}$, and we will see below that we cannot efficiently test all of them—at least not in the classical case. In the *quantum* case, however, we can.

▶ **Theorem 4.** *There is a quantum tester for $\mathcal{P}_{\sqrt{n}/4,\sqrt{n}/2}$ using $O(1)$ queries (and polylog(n) time), while for every even integer $r \in [2, n/2)$, every classical tester for $\mathcal{P}_{r/2,r}$ makes $\Omega(\sqrt{r/\log r \log n})$ queries. In particular, testing $\mathcal{P}_{\sqrt{n}/4,\sqrt{n}/2}$ requires $\Omega(n^{1/4}/\log n)$ classical queries.*

The quantum upper bound is obtained by a small modification of Shor's algorithm: use Shor to find the period (if there is one) and then test this purported period with another $O(1)$ queries.[3] The classical lower is based on ideas from Lachish and Newman [27], who proved classical testing lower bounds for more general periodicity-testing problems. However, while we follow their general outline, we need to modify their proof since it specifically applies to functions with range $\{0, 1\}$, which is different from our 1-1 case. The requirement of being 1-1 within each period is crucial for the upper bound—quantum algorithms need about $\sqrt{n}$ queries to find the period of functions with range $\{0, 1\}$. While our separation is slightly weaker than Aaronson's separation for Fourier checking (our classical lower bound is $n^{1/4}/\log n$ instead $n^{1/4}$), the problem of periodicity testing is arguably more natural, and it may have more applications than Fourier checking.

2 Preliminaries

For any distribution $\mathcal{P}$ on $[m]$ we denote by $\mathcal{P}(j)$ the probability mass of $j \in [m]$ and for any $M \subseteq [m]$ we denote by $\mathcal{P}(M)$ the sum $\sum_{j \in M} \mathcal{P}(j)$. For a function $f : [n] \to [m]$, we denote by $\mathcal{P}_f$ the distribution over $[m]$ in which the weight $\mathcal{P}_f(j)$ of every $j \in [m]$ is proportional to the number of elements $i \in [n]$ that are mapped to j. Formally, for all $j \in [m]$ we define $\mathcal{P}_f(j) \triangleq \Pr_{i \sim U}[f(i) = j] = \frac{|f^{-1}(j)|}{n}$, where U is the uniform distribution on $[n]$, that is $U(i) = 1/n$ for all $i \in [n]$. Whenever the domain is clear from context (and may be something other than $[n]$), we also use U to denote the uniform distribution on that domain.

[3] After a first version of this paper was written, Pranab Sen pointed out to us that the ingredients for our quantum upper bound are already present in work of Hales and Hallgren [24], and in Hales's PhD thesis [23]. However, as also pointed out in the introduction of [19], their results are not stated in the context of property testing. Moreover, no classical lower bounds are proved there; to the best of our knowledge, our lower bound is new.

Let $\|\cdot\|_1$ and $\|\cdot\|_\infty$ stand for L_1-norm and L_∞-norm respectively. Two distributions $\mathcal{P}_f, \mathcal{P}_g$ defined by functions $f, g : [n] \to [m]$ are ϵ-*far* if the L_1-distance between them is at least ϵ. Namely, $\mathcal{P}_f$ is ϵ-far from $\mathcal{P}_g$ if $\|\mathcal{P}_f - \mathcal{P}_g\|_1 = \sum_{j=1}^{m} |\mathcal{P}_f(j) - \mathcal{P}_g(j)| \geq \epsilon$.

2.1 Bucketing

Bucketing is a general tool, introduced in [8, 7], that decomposes any explicitly given distribution into a collection of distributions that are almost uniform. In this section we recall the bucketing technique and the lemmas (from [8, 7]) that we will need for our proofs.

▶ **Definition 5.** Given a distribution $\mathcal{P}$ over $[m]$, and $M \subseteq [m]$ such that $\mathcal{P}(M) > 0$, the *restriction* $\mathcal{P}_{|M}$ is a distribution over M with $\mathcal{P}_{|M}(i) = \mathcal{P}(i)/\mathcal{P}(M)$.

Given a partition $\mathcal{M} = \{M_0, M_1, \ldots, M_k\}$ of $[m]$, we denote by $\mathcal{P}_{\langle \mathcal{M} \rangle}$ the distribution over $\{0\} \cup [k]$ in which $\mathcal{P}_{\langle \mathcal{M} \rangle}(i) = \mathcal{P}(M_i)$.

Given an explicit distribution $\mathcal{P}$ over $[m]$, $Bucket(\mathcal{P}, [m], \epsilon)$ is a procedure that generates a partition $\{M_0, M_1, \ldots, M_k\}$ of the domain $[m]$, where $k = \frac{2 \log m}{\log(1+\epsilon)}$. This partition satisfies the following conditions:

- $M_0 = \{j \in [m] \mid \mathcal{P}(j) < \frac{1}{m \log m}\}$;
- for all $i \in [k]$, $M_i = \left\{ j \in [m] \mid \frac{(1+\epsilon)^{i-1}}{m \log m} \leq \mathcal{P}(j) < \frac{(1+\epsilon)^i}{m \log m} \right\}$.

▶ **Lemma 6** ([7]). *Let $\mathcal{P}$ be a distribution over $[m]$ and let $\{M_0, M_1, \ldots, M_k\} \leftarrow Bucket(\mathcal{P}, [m], \epsilon)$. Then (i) $\mathcal{P}(M_0) \leq 1/\log m$; (ii) for all $i \in [k]$, $\|\mathcal{P}_{|M_i} - U_{|M_i}\|_1 \leq \epsilon$.*

▶ **Lemma 7** ([7]). *Let $\mathcal{P}, \mathcal{P}'$ be two distributions over $[m]$ and let $\mathcal{M} = \{M_0, M_1, \ldots, M_k\}$ be a partition of $[m]$. If $\|\mathcal{P}_{|M_i} - \mathcal{P}'_{|M_i}\|_1 \leq \epsilon_1$ for every $i \in [k]$ and if in addition $\|\mathcal{P}_{\langle \mathcal{M} \rangle} - \mathcal{P}'_{\langle \mathcal{M} \rangle}\|_1 \leq \epsilon_2$, then $\|\mathcal{P} - \mathcal{P}'\|_1 \leq \epsilon_1 + \epsilon_2$.*

▶ **Corollary 8.** *Let $\mathcal{P}, \mathcal{P}'$ be two distributions over $[m]$ and let $\mathcal{M} = \{M_0, M_1, \ldots, M_k\}$ be a partition of $[m]$. If $\|\mathcal{P}_{|M_i} - \mathcal{P}'_{|M_i}\|_1 \leq \epsilon_1$ for every $i \in [k]$ such that $\mathcal{P}(M_i) \geq \epsilon_3/k$, and if in addition $\|\mathcal{P}_{\langle \mathcal{M} \rangle} - \mathcal{P}'_{\langle \mathcal{M} \rangle}\|_1 \leq \epsilon_2$, then $\|\mathcal{P} - \mathcal{P}'\|_1 \leq 2(\epsilon_1 + \epsilon_2 + \epsilon_3)$.*

2.2 Quantum Queries and Approximate Counting

Since we only use specific quantum procedures as a black-box in otherwise classical algorithms, we will not explain the model of quantum query algorithms in much detail (see [28, 13] for that). Suffice it to say that the function f is assumed to be accessible by the oracle unitary transformation O_f, which acts on a $(\log n + \log m)$-qubit space by sending the basis vector $|x\rangle|b\rangle$ to $|x\rangle|b \oplus f(x)\rangle$ where $\oplus$ is bitwise addition modulo two.

The following lemma allows us to estimate the size of the pre-image of a set $S \subseteq [m]$ under f. It follows easily from the work of Brassard, Høyer, Mosca, and Tapp [10, Theorem 13].

▶ **Lemma 9.** *For every $\delta \in [0, 1]$, for every oracle O_f for the function $f : [n] \to [m]$, and for every set $S \subseteq [m]$, there is a quantum algorithm* QEstimate(f, S, δ) *that makes $O(m^{1/3}/\delta)$ queries to f and, with probability at least $5/6$, outputs an estimate p' to $p = \mathcal{P}_f(S) = |f^{-1}(S)|/n$ such that $|p' - p| \leq \frac{\delta\sqrt{p}}{m^{1/3}} + \frac{\delta^2}{m^{2/3}}$.*

3 Proof of Theorem 1

3.1 Testing Uniformity Tolerantly

Given $\epsilon > 0$ and oracle access to a function $f : [n] \to [m]$, our task is to distinguish the case $\|\mathcal{P}_f - U\|_1 \geq \epsilon$ from the case $\|\mathcal{P}_f - U\|_\infty \leq \epsilon/4m$. Note that this is a stronger condition

than the one required for the usual testing task, where the goal is to distinguish the case $\|\mathcal{P}_f - U\|_1 \geq \epsilon$ from $\|\mathcal{P}_f - U\|_\infty = \|\mathcal{P}_f - U\|_1 = 0$.

▶ **Theorem 10.** *There is a quantum testing algorithm (Algorithm 1, below) that given $\epsilon > 0$ and oracle access to a function $f : [n] \to [m]$ makes $O(\frac{m^{1/3}}{\epsilon^2})$ quantum queries and with probability at least $2/3$ outputs REJECT if $\|\mathcal{P}_f - U\|_1 \geq \epsilon$, and ACCEPT if $\|\mathcal{P}_f - U\|_\infty \leq \epsilon/4m$.*

Algorithm 1 (Tests closeness to the uniform distribution.)

pick a set $T \subseteq [n]$ of $t = m^{1/3}$ indices uniformly at random
query f on all indices in T; set $S \leftarrow \{f(i) \mid i \in T\}$
if $f(i) = f(j)$ for some $i, j \in T$, $i \neq j$ (or equivalently, $|S| < t$) **then**
  REJECT
end if
$p' \leftarrow \text{QEstimate}(f, S, \delta)$, with $\delta \triangleq \frac{\epsilon^2}{320}$
if $|p' - \frac{t}{m}| \leq 32\delta\frac{t}{m}$ **then**
  ACCEPT
else
  REJECT
end if

We need the following corollary for the actual application of Theorem 10:

▶ **Corollary 11.** *There is an "amplified" version of Algorithm 1 that given $\epsilon > 0$ and oracle access to a function $f : [n] \to [m]$ makes $O(\frac{m^{1/3}\log\log m}{\epsilon^2})$ quantum queries and with probability at least $1 - \frac{1}{\log^2 m}$ outputs REJECT if $\|\mathcal{P}_f - U\|_1 \geq \epsilon$, and ACCEPT if $\|\mathcal{P}_f - U\|_\infty \leq \epsilon/4m$.*

of Theorem 10. Notice that Algorithm 1 makes only $O(\frac{m^{1/3}}{\epsilon^2})$ queries: $t = m^{1/3}$ classical queries are made initially, and the call to QEstimate requires additional $O(m^{1/3}/\delta) = O(\frac{m^{1/3}}{\epsilon^2})$ queries.

Now we show that Algorithm 1 satisfies the correctness conditions in Theorem 10. Let $V \subseteq [m]$ denote the multi-set of values $\{f(x) \mid x \in T\}$ (unlike S, the multi-set V may contain some element of $[m]$ more than once). If $\|\mathcal{P}_f - U\|_\infty \leq \epsilon/4m$ then $\mathcal{P}_f(V) \leq (1 + \frac{\epsilon}{4})t/m$, and hence

$$p(t;m) \triangleq \Pr[\text{the elements in } V \text{ are distinct}] \geq \left(1 - \frac{(1+\frac{\epsilon}{4})t}{m}\right)^t \geq 1 - \frac{(1+\frac{\epsilon}{4})t^2}{m} > 1 - o(1).$$

Thus if $\|\mathcal{P}_f - U\|_\infty \leq \epsilon/4m$ then with probability at least $1 - o(1)$, the tester does not discover any collision. If, on the other hand, $\|\mathcal{P}_f - U\|_1 \geq \epsilon$ and a collision is discovered, then the tester outputs REJECT, as expected. Hence the following lemma suffices for completing the proof of Theorem 10.

▶ **Lemma 12.** *Conditioned on the event that all elements in V are distinct, we have*

- *if $\|\mathcal{P}_f - U\|_\infty \leq \epsilon/4m$ then $\Pr\left[|\mathcal{P}_f(V) - t/m| \leq \frac{3\epsilon^2 t}{32m}\right] \geq 1 - o(1)$;*
- *if $\|\mathcal{P}_f - U\|_1 \geq \epsilon$ then $\Pr\left[|\mathcal{P}_f(V) - t/m| > \frac{3\epsilon^2 t}{16m}\right] \geq 1 - o(1)$.*

We omit the proof of Lemma 12 from this extended abstract. Assuming Lemma 12, we first prove Theorem 10. Set $p \triangleq \mathcal{P}_f(V)$, and recall that $t/m = 1/m^{2/3}$.

If $\|\mathcal{P}_f - U\|_\infty \leq \epsilon/4m$ then with probability at least $1-o(1)$ the elements in V are distinct and also $|p - 1/m^{2/3}| \leq \frac{30\delta}{m^{2/3}}$. In this case, by Lemma 9, with probability at least 5/6 the estimate p' computed by QEstimate satisfies $|p-p'| \leq \frac{\delta\sqrt{p}}{m^{1/3}} + \frac{\delta^2}{m^{2/3}} \leq \frac{\delta\sqrt{(1+30\delta)/m^{2/3}}}{m^{1/3}} + \frac{\delta^2}{m^{2/3}} \leq \frac{2\delta}{m^{2/3}}$, and by the triangle inequality $|p' - \frac{t}{m}| \leq 32\delta\frac{t}{m}$. Hence the overall probability that Algorithm 1 outputs ACCEPT is at least $5/6 - o(1) > 2/3$.

If $\|\mathcal{P}_f - U\|_1 \geq \epsilon$, then either Algorithm 1 discovers a collision and outputs REJECT, or otherwise, $|p - 1/m^{2/3}| > \frac{60\delta}{m^{2/3}}$ with probability $1 - o(1)$. In the latter case, we make the following case distinction.

- **Case** $p \leq 10/m^{2/3}$**:** By Lemma 9, with probability at least 5/6 the estimate p' of QEstimate satisfies $|p - p'| \leq \frac{\delta\sqrt{p}}{m^{1/3}} + \frac{\delta^2}{m^{2/3}} < \frac{10\delta}{m^{2/3}}$. Then by the triangle inequality, $|p' - \frac{t}{m}| > \frac{60\delta}{m^{2/3}} - \frac{10\delta}{m^{2/3}} > 32\delta\frac{t}{m}$.
- **Case** $p > 10/m^{2/3}$**:** In this case it is sufficient to prove that with probability at least 5/6, $p' \geq p/2$ (which clearly implies $|p' - \frac{t}{m}| > 32\delta\frac{t}{m}$). This follows again by Lemma 9, since $p > 10/m^{2/3}$ implies $\frac{\delta\sqrt{p}}{m^{1/3}} + \frac{\delta^2}{m^{2/3}} \leq p/2$.

So the overall probability that Algorithm 1 outputs REJECT is at least $5/6 - o(1) > 2/3$. ◀

3.2 Testing Closeness to a Known Distribution

In this section we prove Theorem 1 based on Theorem 10. Let $\mathcal{P}_f$ be an unknown distribution and let $\mathcal{P}_g$ be a known distribution, defined by $f, g : [n] \to [m]$ respectively. We show that for any $\epsilon > 0$, Algorithm 2 makes $O(\frac{m^{1/3}\log^2 m \log\log m}{\epsilon^5})$ queries and distinguishes the case $\|\mathcal{P}_f - \mathcal{P}_g\|_1 = 0$ from the case $\|\mathcal{P}_f - \mathcal{P}_g\|_1 > 5\epsilon$ with probability $\geq 2/3$, satisfying the requirements of Theorem 1.

Algorithm 2 (Tests closeness to a known distribution.)

let $\mathcal{M} \triangleq \{M_0, \ldots, M_k\} \leftarrow Bucket(\mathcal{P}_g, [m], \frac{\epsilon}{4})$ for $k = \frac{2\log m}{\log(1+\epsilon/4)}$
for $i = 1$ to k **do**
  if $\mathcal{P}_g(M_i) \geq \epsilon/k$ **then**
    if $\|(\mathcal{P}_f)_{|M_i} - U_{|M_i}\|_1 \geq \epsilon$ (check using the amplified version of Algorithm 1 from Corollary 11) **then**
      REJECT
    end if
  end if
end for
if $\|(\mathcal{P}_f)_{\langle\mathcal{M}\rangle} - (\mathcal{P}_g)_{\langle\mathcal{M}\rangle}\|_1 > \epsilon/4$ (check classically with $O(\sqrt{k}) = O(\log m)$ queries [7]) **then**
  REJECT
end if
ACCEPT

Observe that no queries are made by Algorithm 2 itself, and the total number of queries made by calls to Algorithm 1 is bounded by $k \cdot O(\frac{k}{\epsilon} \cdot \frac{m^{1/3}\log\log m}{\epsilon^2}) + O(\sqrt{k}) = O(\frac{m^{1/3}\log^2 m \log\log m}{\epsilon^5})$.[4] In addition, the failure probability of Algorithm 1 is at most

[4] The additional factor of $\frac{k}{\epsilon}$ is for executing Algorithm 1 on the conditional distributions $(\mathcal{P}_f)_{|M_i}$, with $\mathcal{P}_f(M_i) \geq \frac{\epsilon}{k}$.

$1/\log^2 m \ll 1/k$, so we can assume that with high probability none of its executions failed.

For any $i \in [k]$ and any $x \in M_i$, by the definition of the buckets $\frac{(1+\epsilon/4)^{i-1}}{m \log m} \leq \mathcal{P}_g(x) \leq \frac{(1+\epsilon/4)^i}{m \log m}$. Thus, for any $i \in [k]$ and $x \in M_i$, $(1-\frac{\epsilon}{4})/|M_i| < 1/(1+\frac{\epsilon}{4})|M_i| < (\mathcal{P}_g)_{|M_i}(x) < (1+\frac{\epsilon}{4})/|M_i|$, or equivalently for any $i \in [k]$ we have $\|(\mathcal{P}_g)_{|M_i} - U_{|M_i}\|_\infty \leq \frac{\epsilon}{4|M_i|}$. This means that if $\|\mathcal{P}_f - \mathcal{P}_g\|_1 = 0$ then

1. for any $i \in [k]$, $\|(\mathcal{P}_f)_{|M_i} - U_{|M_i}\|_\infty \leq \frac{\epsilon}{4|M_i|}$ and thus the tester never outputs REJECT in Line 5 (since we assumed that Algorithm 1 did not err in any of its executions).
2. $\|(\mathcal{P}_f)_{\langle \mathcal{M} \rangle} - (\mathcal{P}_g)_{\langle \mathcal{M} \rangle}\|_1 = 0$, and hence the tester does not output REJECT in Line 10 either.

On the other hand, if $\|\mathcal{P}_f - \mathcal{P}_g\|_1 > 5\epsilon$ then by Corollary 8 either $|(\mathcal{P}_f)_{\langle \mathcal{M} \rangle} - (\mathcal{P}_g)_{\langle \mathcal{M} \rangle}| > \epsilon/4$ or there is at least one $i \in [k]$ for which $\mathcal{P}_f(M_i) \geq \epsilon/k$ and $\|(\mathcal{P}_f)_{|M_i} - (\mathcal{P}_g)_{|M_i}\|_1 > 5\epsilon/4$ (otherwise $\|\mathcal{P}_f - \mathcal{P}_g\|_1$ must be smaller than $2(5\epsilon/4 + \epsilon/4 + \epsilon) = 5\epsilon$). In the first case the tester will reject in Line 10. In the second case the tester will reject in Line 5 as $\|(\mathcal{P}_f)_{|M_i} - (\mathcal{P}_g)_{|M_i}\|_1 > 5\epsilon/4$ implies (by the triangle inequality) $\|(\mathcal{P}_f)_{|M_i} - U_{|M_i}\|_1 > \epsilon$, since $\|(\mathcal{P}_g)_{|M_i} - U_{|M_i}\|_1 < \epsilon/4$ by Lemma 6.

4 Proof of Theorem 4

4.1 Quantum Upper Bound

The quantum tester is very simple, and completely based on existing ideas. First, run a variant of Shor's algorithm to find the period of f (if there is one), using $O(1)$ queries. Second, test whether the purported period is indeed the period, using another $O(1)$ queries as described above. Accept iff the latter test accepts.

For the sake of completeness we sketch here how Shor's algorithm can be used to find the unknown period p of an f that is promised to be 1-1-p-periodic for some value of $p \leq \sqrt{n}/2$. Here is the algorithm:

1. First prepare the 2-register quantum state $\frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|0\rangle$
2. Query f once (in superposition), giving $\frac{1}{\sqrt{n}} \sum_{i \in [n]} |i\rangle|f(i)\rangle$
3. Measure the second register, which gives some $f(s)$ for $s \in [p]$ and collapses the first register to the i having the same f-value: $\frac{1}{\sqrt{\lfloor n/p \rfloor}} \sum_{i \in [n], i=s \bmod p} |i\rangle|f(i)\rangle$
4. Do a quantum Fourier transform on the first register and measure.
   Some analysis shows that with high probability the measurement gives an i such that $\left|\frac{i}{n} - \frac{c}{p}\right| < \frac{1}{2n}$, where c is a random (essentially uniform) integer in $[p]$. Using continued fraction expansion, we can then calculate the unknown fraction c/p from the known fraction i/n.[5]

[5] Two distinct fractions each with denominator $\leq \sqrt{n}/2$ are $\geq 4/n$ apart. Hence there is only one fraction with denominator at most $\sqrt{n}/2$ within distance $2/n$ from the known fraction i/n. This unique fraction can only be c/p, and CFE efficiently finds it for us. Note that we do not obtain c and p separately, but just their ratio given as a numerator and a denominator in lowest terms. If c and p were coprime that would be enough, but that need not happen with high probability.

5. Doing the above 4 steps k times gives fractions $c_1/p, \ldots, c_k/p$, each given as a numerator and a denominator (in lowest terms). Each of the k denominators divides p, and if k is a sufficiently large constant then with high probability (over the c_i's), their least common multiple is p.

4.2 Classical Lower Bound

We saw above that quantum computers can efficiently test 1-1-PERIODICITY $\mathcal{P}_{\sqrt{n}/4,\sqrt{n}/2}$. Here we will show that this is not the case for classical testers: those need roughly $\sqrt{r}$ queries for 1-1-periodicity testing $\mathcal{P}_{r/2,r}$, in particular roughly $n^{1/4}$ queries for $r = \sqrt{n}/2$. Our proof follows along the lines of Lachish and Newman [27]. However, since their proof applies to functions with range 0/1 that need not satisfy the 1-1 property, some modifications are needed.

Fix a sufficiently large even integer $r < n/2$. We will use Yao's principle, proving a lower bound for *deterministic* query testers with error probability $\leq 1/3$ in distinguishing two distributions, one on negative instances and one on positive instances. First, the "negative" distribution $\mathcal{D}_N$ is uniform on all $f : [n] \to [m]$ that are ϵ-far from $\mathcal{P}_{r/2,r}$. Second, the "positive" distribution $\mathcal{D}_P$ chooses a *prime* period $p \in [r/2, r]$ uniformly, then chooses a 1-1 function $[p] \to [m]$ uniformly (equivalently, chooses a sequence of p distinct elements from $[m]$), and then completes f by repeating this period until the domain $[n]$ is "full". Note that the last period will not be completed if $p \not| n$.

Suppose $q = o(\sqrt{r/\log r \log n})$ is the number of queries of our deterministic tester. Fix a set $Q = \{i_1, \ldots, i_q\} \subseteq [n]$ of q queries. Let $f(Q) \in [m]^q$ denote the concatenated answers $f(i_1), \ldots, f(i_q)$. We prove two lemmas, one for the negative and one for the positive distribution, showing $f(Q)$ to be close to uniformly distributed in both cases. Both the proofs are omitted from this extended abstract.

▶ **Lemma 13.** *For all $\eta \in [m]^q$, we have $\Pr_{\mathcal{D}_N}[f(Q) = \eta] = (1 \pm o(1))m^{-q}$.*

▶ **Lemma 14.** *There exists an event B such that $\Pr_{\mathcal{D}_P}[B] = o(1)$, and for all $\eta \in [m]^q$ with distinct coordinates, we have $\Pr_{\mathcal{D}_P}[f(Q) = \eta \mid \overline{B}] = (1 \pm o(1))m^{-q}$.*

Since $(1 - o(1))m^q$ of all $\eta \in [m]^q$ have distinct coordinates, their weight under $\mathcal{D}_P$ sums to $1 - o(1)$, and the other possible η comprise only a $o(1)$-fraction of the overall weight. The query-answers $f(Q)$ are the only access the algorithm has to the input. Hence the previous two lemmas imply that an algorithm with $o(\sqrt{r/\log r \log n})$ queries cannot distinguish $\mathcal{D}_P$ and $\mathcal{D}_N$ with probability better than $1/2 + o(1)$. This establishes the claimed classical lower bound.

5 Summary and Open Problems

In this paper we studied and compared the quantum and classical query complexities of a number of testing problems. The first problem is deciding whether two probability distributions on a set $[m]$ are equal or ϵ-far. Our main result is a quantum tester for the case where one of the two distributions is known (i.e., given explicitly) while the other is unknown and represented by a function that can be queried. Our tester uses roughly $m^{1/3}$ queries to the function, which is essentially optimal. It would be very interesting to extend this quantum upper bound to the case where *both* distributions are unknown. Such a quantum tester would show that the known-unknown and unknown-unknown cases have the same complexity in the quantum world. In contrast, they are known to have different complexities

in the classical world: about $m^{1/2}$ queries for the known-unknown case and about $m^{2/3}$ queries for the unknown-unknown case. The classical counterparts of these tasks play an important role in many problems related to property testing. We already mentioned one example, the graph isomorphism problem, where distribution testers are used as a black-box. We hope that the quantum analogues developed here and in [11] will find similar use.

The second testing problem is deciding whether a given function $f : [n] \to [m]$ is periodic or far from periodic. For the specific version of the problem that we considered (where in the first case the period is at most about $\sqrt{n}$, and the function is injective within each period), we proved that quantum testers need only a constant number of queries (using Shor's algorithm), while classical algorithms need about $n^{1/4}$ queries. Both this result and Aaronson's recent result on "Fourier checking" [1] contrast with the constant-vs-$\log n$ and $\log n$-vs-$\sqrt{n}$ separations obtained by Buhrman et al. [12] for other testing problems, but still leave open their question: is there a testing problem where the separation is "maximal", in the sense that quantum testers need only $O(1)$ queries while classical testers need $\Omega(n)$?

Acknowledgements We thank Avinatan Hassidim, Harry Buhrman and Prahladh Harsha for useful discussions, Frederic Magniez for a reference to [19], Pranab Sen for a reference to [24, 23], and Scott Aaronson for pointing out that his Fourier checking result in [1] was the first constant-vs-polynomial quantum speed-up in property testing.

References

1 S. Aaronson. BQP and the Polynomial Hierarchy. In *Proceedings of 42nd ACM STOC*, 2010. arXiv:0910.4698.

2 S. Aaronson and Y. Shi. Quantum lower bounds for the collision and the element distinctness problems. *Journal of the ACM*, 51(4):595–605, 2004.

3 A. Ambainis. Polynomial degree and lower bounds in quantum complexity: Collision and element distinctness with small range. *Theory of Computing*, 1(1):37–46, 2005. quant-ph/0305179.

4 A. Ambainis. Quantum walk algorithm for element distinctness. *SIAM Journal on Computing*, 37(1):210–239, 2007. Earlier version in FOCS'04. quant-ph/0311001.

5 A. Ambainis, A. Childs, B. Reichardt, R. Špalek, and S. Zhang. Any AND-OR formula of size n can be evaluated in time $N^{1/2+o(1)}$ on a quantum computer. In *Proceedings of 48th IEEE FOCS*, 2007.

6 A. Atici and R. Servedio. Quantum algorithms for learning and testing juntas. *Quantum Information Processing*, 6(5):323–348, 2009.

7 T. Batu, L. Fortnow, E. Fischer, R. Kumar, R. Rubinfeld, and P. White. Testing random variables for independence and identity. In *Proceedings of 42nd IEEE FOCS*, pages 442–451, 2001.

8 T. Batu, L. Fortnow, R. Rubinfeld, W. D. Smith, and P. White. Testing that distributions are close. In *Proceedings of 41st IEEE FOCS*, pages 259–269, 2000.

9 M. Blum, M. Luby, and R. Rubinfeld. Self-testing/correcting with applications to numerical problems. *Journal of Computer and System Sciences*, 47(3):549–595, 1993. Earlier version in STOC'90.

10 G. Brassard, P. Høyer, M. Mosca, and A. Tapp. Quantum amplitude amplification and estimation. In *Quantum Computation and Quantum Information: A Millennium Volume*, volume 305 of *AMS Contemporary Mathematics Series*, pages 53–74. 2002. quant-ph/0005055.

11 S. Bravyi, A. Hassidim, and A. Harrow. Quantum algorithms for testing properties of distributions. In *Proceedings of 27th Annual Symposium on Theoretical Aspects of Computer Science (STACS'2010)*, 2010. abs/0907.3920.

12 H. Buhrman, L. Fortnow, I. Newman, and H. Röhrig. Quantum property testing. In *Proceedings of 14th ACM-SIAM SODA*, pages 480–488, 2003. quant-ph/0201117.

13 H. Buhrman and R. de Wolf. Complexity measures and decision tree complexity: A survey. *Theoretical Computer Science*, 288(1):21–43, 2002.

14 A. Childs and Y-K. Liu. Quantum algorithms for testing bipartiteness and expansion of bounded-degree graphs. Manuscript, Oct 22, 2009.

15 D. Deutsch and R. Jozsa. Rapid solution of problems by quantum computation. In *Proceedings of the Royal Society of London*, volume A439, pages 553–558, 1992.

16 E. Farhi, J. Goldstone, and S. Gutmann. A quantum algorithm for the Hamiltonian NAND tree. *Theory of Computing*, 4(1):169–190, 2008. quant-ph/0702144.

17 E. Fischer and A. Matsliah. Testing graph isomorphism. *SIAM Journal on Computing*, 38(1):207–225, 2008.

18 Eldar Fischer. The art of uninformed decisions. *Bulletin of the EATCS*, 75:97, 2001.

19 K. Friedl, F. Magniez, M. Santha, and P. Sen. Quantum testers for hidden group properties. *Fundamenta Informaticae*, 91(2):325–340, 2009. Earlier version in MFCS'03.

20 O. Goldreich, S. Goldwasser, and D. Ron. Property testing and its connection to learning and approximation. *Journal of the ACM*, 45(4):653–750, 1998.

21 O. Goldreich and D. Ron. On testing expansion in bounded-degree graphs. *Electronic Colloquium on Computational Complexity (ECCC)*, 7(20), 2000.

22 L. K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of 28th ACM STOC*, pages 212–219, 1996. quant-ph/9605043.

23 L. Hales. *The Quantum Fourier Transform and Extensions of the Abelian Hidden Subgroup Problem*. PhD thesis, University of California, Berkeley, 2002. quant-ph/0212002.

24 L. Hales and S. Hallgren. An improved quantum Fourier transform algorithm and applications. In *Proceedings of 41st IEEE FOCS*, pages 515–525, 2000.

25 Y. Inui and F. Le Gall. Quantum property testing of group solvability. In *Proceedings of 8th LATIN*, pages 772–783, 2008.

26 S. Kutin. Quantum lower bound for the collision problem with small range. *Theory of Computing*, 1(1):29–36, 2005. quant-ph/0304162.

27 O. Lachish and I. Newman. Testing periodicity. *Algorithmica*, 2009. Earlier version appeared in RANDOM'05.

28 M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

29 D. Ron. Property testing: A learning theory perspective. *Foundations and Trends in Machine Learning*, 1(3):307–402, 2008.

30 P. W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997. Earlier version in FOCS'94. quant-ph/9508027.

31 D. Simon. On the power of quantum computation. *SIAM Journal on Computing*, 26(5):1474–1483, 1997. Earlier version in FOCS'94.

32 P. Valiant. Testing symmetric properties of distributions. In *Proceedings of 40th ACM STOC*, pages 383–392, 2008.

Lower bounds for Quantum Oblivious Transfer*

André Chailloux[1], Iordanis Kerenidis[1], and Jamie Sikora[2]

1 **LRI, Univ Paris-Sud**
CNRS
{chaillou,jkeren}@lri.fr
2 **IQC**
University of Waterloo
jwjsikor@uwaterloo.ca

Abstract

Oblivious transfer is a fundamental primitive in cryptography. While perfect information theoretic security is impossible, quantum oblivious transfer protocols can limit the dishonest players' cheating. Finding the optimal security parameters in such protocols is an important open question. In this paper we show that every 1-out-of-2 oblivious transfer protocol allows a dishonest party to cheat with probability bounded below by a constant strictly larger than 1/2. Alice's cheating is defined as her probability of guessing Bob's index, and Bob's cheating is defined as his probability of guessing both input bits of Alice. In our proof, we relate these cheating probabilities to the cheating probabilities of a coin flipping protocol and conclude by using Kitaev's coin flipping lower bound. Then, we present an oblivious transfer protocol with two messages and cheating probabilities at most 3/4. Last, we extend Kitaev's semidefinite programming formulation to more general primitives, where the security is against a dishonest player trying to force the outcome of the other player, and prove optimal lower and upper bounds for them.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.157

1 Introduction

Quantum information enables us to do cryptography with information theoretic security. The first breakthrough result in quantum cryptography is the unconditionally secure key distribution protocol of Bennett and Brassard [BB84]. Since then, a long series of work has studied which other cryptographic primitives are possible in the quantum world. However, the subsequent results were negative. Mayers and Lo, Chau proved the impossibility of secure ideal quantum bit commitment and oblivious transfer and consequently of any type of two-party secure computation [May97, LC97, DKSW07]. On the other hand, several imperfect variants of these primitives have been shown to be possible. Finding the optimal parameters for such fundamental primitives has been since an important open question. The reason for looking at these abstract primitives is that they are the basis for all cryptographic protocols one may wish to construct, including identification schemes, digital signatures, electronic voting, etc. Let us emphasize that in this paper we only look at information theoretic security and we do not discuss computational security or security in restricted models like the bounded-storage or noisy-storage model.

We start with coin flipping, which was first proposed by Blum [Blu81] and has since found numerous applications in two-party secure computation. Even though the results of Mayers

* This work was partially supported by the projects ANR-09-JCJC-0067-01, ANR-08-EMER-012, CSQIP EU-Canada Collaboration, NSERC, MITACS, and ERA (Ontario)

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 157–168

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

and of Lo and Chau exclude the possibility of perfect quantum coin flipping, i.e., where the resulting coin is perfectly unbiased, it still remained open whether one can construct a quantum protocol where no player could bias the coin with probability 1. Aharonov et al. [ATVY00] provided such a protocol where no dishonest player could bias the coin with probability higher than 0.9143. Then, Ambainis [Amb01] described an improved protocol whose cheating probability was at most 3/4. Subsequently, a number of different protocols had been proposed [SR01, NS03, KN04] that achieved the same bound of 3/4.

On the other hand, Kitaev [Kit03], using a formulation of quantum coin flipping as semidefinite programs proved a lower bound of 1/2 on the product of the cheating probabilities for Alice and Bob (see [ABDR04]). In other words, no quantum coin flipping protocol can achieve a cheating probability less than $1/\sqrt{2}$ for both Alice and Bob.

The question of whether 3/4 or $1/\sqrt{2}$ was the right answer has recently been resolved by Chailloux and Kerenidis [CK09] who described a protocol with cheating probability arbitrarily close to $1/\sqrt{2}$. In their protocol they use as a subroutine a weaker variant of coin flipping which is referred to as *weak coin flipping*.

In this paper, we focus on oblivious transfer, which is a universal primitive for any two-party secure computation [Rab81, EGL82, Cré87]. We define a 1-out-of-2 random oblivious transfer protocol with bias ε, denoted here as random-OT.

The first impossibility result for quantum OT with information theoretic security was shown by Lo [Lo97]. However, not much was known about the best possible bias that one can get for OT. Note that Kitaev's lower bound does not a priori hold for OT, since we do not know how to easily convert an OT protocol to a coin flipping protocol without any loss.

In related work, Salvail, Schaffner and Sotakova [SSS09] have quantitatively studied a different notion of security for OT protocols (and generally any two-party protocols) that they call information leakage. They prove, among other results, that any 1-out-of-2 OT protocol has a constant leakage. Their model is somewhat different, for example they do not allow the players to abort during the protocol, and their security notion is described in terms of mutual information and entropy and does not immediately translate to our security notion of guessing probabilities. However, their results provide more evidence that almost-perfect OT protocols are impossible for different variants of security.

In another work, Jain, Radhakrishnan and Sen [JRS02] showed that in a 1-out-of-n OT protocol, if Alice gets t bits of information about Bob's index b, then Bob gets at least $\Omega(n/2^{O(t)})$ bits of information about Alice's string x.

Our work

In this paper, we quantitatively study the bias of quantum oblivious transfer protocols. More precisely, we construct a coin flipping protocol that uses OT as a subroutine and show a relation between the cheating probabilities of the OT protocol and the ones of the coin flipping protocol. Then, using Kitaev's lower bound for coin flipping we derive a non-trivial lower bound (albeit weaker) on the cheating probabilities for OT. More precisely we prove the following theorem.

▶ **Theorem 1.** *In any quantum oblivious transfer protocol, we have*

$$A_{OT} \cdot f(B_{OT}) \geq 1/2$$

where f is a function that we define later[1]*. This implies for the bias ε of the protocol that*

$$\varepsilon \geq \frac{1}{2}\left(\sqrt{\frac{1}{2}+2\sqrt{2}}-\sqrt{\frac{1}{2}}\right)-\frac{1}{2} \approx 0.0586.$$

Moreover, in Section 4 we describe a simple 1-out-of-2 random-OT protocol and analyze the cheating probabilities of Alice and Bob.

▶ **Theorem 2.** *There exists a quantum oblivious transfer protocol such that $A_{OT} = B_{OT} = \frac{3}{4}$.*

One may wonder if it would be possible to extend Kitaev's semidefinite programming formulation to include the OT primitive and get a stronger lower bound this way. In Section 5 we describe a generalization of Kitaev's semidefinite program that captures a variant of the general k-out-of-n OT primitive. Coin flipping, is then the special case of 1-out-of-1 OT. However, there is a big difference. What the semidefinite program formulation captures is the probability that one player can force the outcome of the other one. More precisely, we define a k-out-of-n forcing oblivious transfer protocol, denoted here as $\binom{n}{k}$-fOT.

We show the following theorem.

▶ **Theorem 3.** *In any $\binom{n}{k}$-fOT protocol and consistent b, x, x_b we have*

$$B_x \cdot A_{b,x_b} \geq \Pr[\textit{Alice honestly outputs } x \textit{ and Bob honestly outputs } (b, x_b)] = \frac{1}{\binom{n}{k}2^n}.$$

In particular, the forcing bias satisfies $\varepsilon \geq \sqrt{2}^k$.

Note that for the special case of coin flipping, or else $\binom{1}{1}$-fOT, our bounds are tight (a multiplicative bias of $\sqrt{2}$ is equivalent to a cheating probability of $\frac{1}{\sqrt{2}}$).

Similar to coin flipping, one can get optimal protocols as well for $\binom{n}{k}$-fOT.

▶ **Theorem 4.** *Let $\gamma > 0$. There exists a protocol for $\binom{n}{k}$-fOT with cheating probabilities:*

$$A_{b,x_b} \leq \frac{\sqrt{2}^k(1+\gamma)}{\binom{n}{k}\cdot 2^k} \quad \textit{and} \quad B_x \leq \frac{\sqrt{2}^k(1+\gamma)}{2^n}.$$

2 Preliminaries

In the literature, many different variants of oblivious transfer have been considered. In this paper, we mainly consider random oblivious transfer. In the full version, we show how this definition is equivalent to other definitions of oblivious transfer with respect to the bias ε.

▶ **Definition 5** (Random Oblivious Transfer)**.** A 1-out-of-2 quantum random oblivious transfer protocol with bias ε, denoted here as random-OT, is a protocol between Alice and Bob such that:

- Alice outputs two bits (x_0, x_1) or Abort and Bob outputs two bits (b, y) or Abort
- If Alice and Bob are honest, they never Abort, $y = x_b$, Alice has no information about b and Bob has no information about $x_{\overline{b}}$. Also, x_0, x_1, b are uniformly random bits
- $A_{OT} := \sup\{\Pr[\text{Alice guesses } b \text{ and Bob does not Abort}]\} = \frac{1}{2} + \varepsilon_A$
- $B_{OT} := \sup\{\Pr[\text{Bob guesses } (x_0, x_1) \text{ and Alice does not Abort}]\} = \frac{1}{2} + \varepsilon_B$

[1] f is the inverse of the function $g(x) = x(2x-1)^2$ on some domain

- The bias of the protocol is defined as $\varepsilon := \max\{\varepsilon_A, \varepsilon_B\}$

where the suprema are taken over all cheating strategies for Alice and Bob.

Note that this definition is slightly different from usual definitions because we want the exact value of the cheating probabilities and not only an upper bound. This is because we consider both lower bounds and upper bounds for *OT* protocols but we could have equivalent results using the standard definitions.

An important issue is that we quantify the security against a cheating Bob as the probability that he can guess (x_0, x_1). One can imagine a security definition where Bob's guessing probability is not for (x_0, x_1), but for example for $x_0 \oplus x_1$ or any other function $f(x_0, x_1)$. Since we are mostly interested in lower bounds, we believe our definition is the most appropriate one, since a lower bound on the probability of guessing (x_0, x_1) automatically yields a lower bound on the probability of guessing any $f(x_0, x_1)$.

Note also that we do not have composability requirements for such protocols. Our main goal here is to get a constant lower bound for the simplest definition of OT, hence making the result as strong as possible. This is why we use the stand-alone definition. This is also the definition that one can relate most easily to the coin flipping protocols, which are also defined in a stand-alone way, e.g., in Kitaev's bound.

We also define quantum (strong) coin flipping.

▶ **Definition 6.** A quantum coin flipping protocol with bias ε, denoted here as CF, is a protocol between Alice and Bob who agree on an output $a \in \{0, 1, \text{Abort}\}$ such that:

- If Alice and Bob are honest then $\Pr[a = 0] = \Pr[a = 1] = \frac{1}{2}$
- $A_{CF} := \sup\{\max\{\Pr[a = 0], \Pr[a = 1]\}\} = \frac{1}{2} + \varepsilon_A$
- $B_{CF} := \sup\{\max\{\Pr[a = 0], \Pr[a = 1]\}\} = \frac{1}{2} + \varepsilon_B$
- The bias of the protocol is defined as $\varepsilon := \max\{\varepsilon_A, \varepsilon_B\}$

where the suprema are taken over all strategies for Alice and Bob.

3 A Lower Bound on Any Oblivious Transfer Protocol

In this section we prove that the bias of any random-*OT* protocol, and hence any *OT* protocol, is bounded below by a constant. We start from a random-*OT* protocol and first show how to construct a coin flipping protocol. Then, we prove a relation between the cheating probabilities of the coin flipping protocol and those in the random-*OT* protocol. Last, we use Kitaev's lower bound for coin flipping to derive a lower bound for any *OT* protocol.

3.1 From Oblivious Transfer to Coin Flipping

Coin Flipping Protocol via random-*OT*

1. Alice and Bob perform the *OT* protocol to create (x_0, x_1) and (b, x_b) respectively. If the *OT* protocol is aborted then so is the coin flipping protocol.
2. Alice sends $c \in_R \{0, 1\}$ to Bob.
3. Bob sends b and x_b to Alice.
4. If x_b from Bob is consistent with Alice's bits then the output of the protocol is $c \oplus b$. Otherwise Alice aborts.

By definition, A_{OT} and B_{OT} denote the optimal cheating probabilities for Alice and Bob in the random-OT protocol and A_{CF} and B_{CF} denote the optimal cheating probabilities for Alice and Bob in the coin flipping protocol. Kitaev's lower bound on coin flipping implies that $A_{CF}B_{CF} \geq 1/2$. We use this inequality to derive an inequality involving A_{OT} and B_{OT}.

▶ **Theorem 1.** *In any quantum oblivious transfer protocol, we have*

$$A_{OT} \cdot f(B_{OT}) \geq 1/2$$

for the function f defined as[2]

$$f(z) = \frac{1}{6}(3\sqrt{3}\sqrt{27z^2 - 2z} + 27z - 1)^{1/3} + \frac{1}{6}(3\sqrt{3}\sqrt{27z^2 - 2z} + 27z - 1)^{-1/3} + 1/3.$$

This implies that the bias ε of the protocol satisfies

$$\varepsilon \geq \frac{1}{2}\left(\sqrt{\frac{1}{2} + 2\sqrt{2}} - \sqrt{\frac{1}{2}}\right) - \frac{1}{2} \approx 0.0586.$$

In what follows we prove the above theorem.

Let $\neg\perp_A^{CF}$ (resp. $\neg\perp_B^{CF}$) denote the event "Alice (resp. Bob) does not abort during the entire coin flipping protocol". Let $\neg\perp_A^{OT}$ (resp. $\neg\perp_B^{OT}$) denote the event "Alice (resp. Bob) does not abort during the random-OT subroutine".

Cheating Alice

By definition, A_{OT} is the optimal probability of Alice guessing b in the random-OT protocol without Bob aborting. Suppose Alice desires to force 0 in the coin flipping protocol (a similar argument can be made if she wants 1). Bob must not abort and Alice must send $c = b$ in her last message. Notice also that in our coin flipping protocol, honest Bob only aborts in the OT subroutine and hence $\neg\perp_B^{OT} \equiv \neg\perp_B^{CF}$. Thus,

$$A_{CF} = \sup\{\Pr[\text{ (Alice sends } c = b) \wedge \neg\perp_B^{CF}]\} = \sup\{\Pr[\text{ (Alice guesses } b) \wedge \neg\perp_B^{OT}]\} = A_{OT}.$$

where the suprema are taken over all possible strategies for Alice.

Cheating Bob

By definition, B_{OT} is the optimal probability of Bob learning both bits in the random-OT protocol without Alice aborting. Thus,

$$\begin{aligned} B_{OT} &= \sup\{\Pr[\text{ (Bob guesses } (x_0, x_1)) \wedge \neg\perp_A^{OT}]\} \\ &= \sup\{\Pr[\neg\perp_A^{OT}] \cdot \Pr[\text{ (Bob guesses } (x_0, x_1))|\neg\perp_A^{OT}]\}. \end{aligned}$$

where the suprema are taken over all strategies for Bob.

If Bob wants to force 0 in the coin flipping protocol (a similar argument works if he wants to force 1), then first, Alice must not abort in the random-OT protocol and second, Bob must send $b = c$ as well as the correct x_c such that Alice does not abort in the last round of the coin flipping protocol. This is equivalent to saying that Bob succeeds if he guesses x_c and Alice does not abort in the random-OT protocol. Since c is chosen by Alice uniformly at random, we can write the probability of Bob cheating as

[2] f is the inverse function of $g(x) = x(2x-1)^2$ on some domain, see the proof for more details.

$$\begin{aligned} B_{CF} &= \max\left\{\frac{1}{2}\Pr[(\text{Bob guesses } x_0)\wedge\neg\perp_A^{OT}]+\frac{1}{2}\Pr[(\text{Bob guesses } x_1)\wedge\neg\perp_A^{OT}]\right\} \\ &= \max\left\{\Pr[\neg\perp_A^{OT}]\cdot\left(\frac{1}{2}\Pr[(\text{Bob guesses } x_0)|\neg\perp_A^{OT}]+\frac{1}{2}\Pr[(\text{Bob guesses } x_1)|\neg\perp_A^{OT}]\right)\right\}. \end{aligned}$$

Notice that we use "max" instead of "sup" above. This is because an optimal strategy exists for every coin flipping protocol. This is a consequence of strong duality in the semidefinite programming formalism of [Kit03], see [ABDR04] for details.

Let us now fix Bob's optimal cheating strategy in the CF protocol. For this strategy, let $p = \Pr[(\text{Bob guesses } x_0)|\neg\perp_A^{OT}]$, $q = \Pr[(\text{Bob guesses } x_1)|\neg\perp_A^{OT}]$ and $a = \frac{p+q}{2}$. Note that wlog, we can assume that Bob's measurements are projective measurements. This can be done by increasing the dimension of Bob's space. Also, Alice has a projective measurement on her space to determine the bits (x_0, x_1).

We use the following lemma to relate B_{CF} and B_{OT}.

▶ **Lemma 1** (Learning-In-Sequence Lemma). *Let $p, q \in [1/2, 1]$. Let Alice and Bob share a joint pure state. Suppose Alice performs a projective measurement $M = \{M_{x_0,x_1}\}_{x_0,x_1\in\{0,1\}}$ on her space to determine the values of (x_0, x_1). Suppose there is a projective measurement $P = \{P_0, P_1\}$ on Bob's space that allows him to guess bit x_0 with probability p and a projective measurement $Q = \{Q_0, Q_1\}$ on his space that allows him to guess bit x_1 with probability q. Then, there exists a measurement on Bob's space that allows him to guess (x_0, x_1) with probability at least $a(2a-1)^2$ where $a = \frac{p+q}{2}$.*

We postpone the proof of this lemma to Subsection 3.2.

We now construct a cheating strategy for Bob for the OT protocol: run the optimal cheating CF strategy and look at Bob's state after step 1 conditioned on $\neg\perp_A^{OT}$. Note that this event happens with nonzero probability in the optimal coin flipping strategy since otherwise the success probability is 0. The optimal CF strategy gives measurements that allow Bob to guess x_0 with probability p and x_1 with probability q. Bob uses these measurements and the procedure of Lemma 1 to guess (x_0, x_1). Let b be the probability he guesses (x_0, x_1). From Lemma 1, we have that $b \geq a(2a-1)^2$. By definition of B_{OT} and B_{CF}, we have:

$$b = \Pr[\text{ (Bob guesses } (x_0,x_1))|\neg\perp_A^{OT}] \leq \frac{B_{OT}}{\Pr[\neg\perp_A^{OT}]} \quad \text{and} \quad a = \frac{B_{CF}}{\Pr[\neg\perp_A^{OT}]}.$$

This gives us

$$\frac{B_{OT}}{\Pr[\neg\perp_A^{OT}]} \geq \frac{B_{CF}}{\Pr[\neg\perp_A^{OT}]}\left(2\frac{B_{CF}}{\Pr[\neg\perp_A^{OT}]}-1\right)^2 \implies B_{OT} \geq B_{CF}\left(2B_{CF}-1\right)^2,$$

where the implication holds since $B_{CF} \geq 1/2$.

We now calculate an upper bound on B_{CF} as a function of B_{OT}. Let $g(x) = x(2x-1)^2$. It can be easily checked that g is bijective on the interval $[0.5, 1]$ and increasing. Let f be the inverse function of g from $[0, 1]$ to $[0, 0.5]$. Since g is increasing, f is also increasing. Hence, since $B_{OT} \geq g(B_{CF})$ and $B_{CF} \in [0.5, 1]$, we conclude that

$$B_{CF} \leq f(B_{OT}).$$

We can write f analytically using computer software to get the following function

$$f(z) = \frac{1}{6}(3\sqrt{3}\sqrt{27z^2 - 2z} + 27z - 1)^{1/3} + \frac{1}{6}(3\sqrt{3}\sqrt{27z^2 - 2z} + 27z - 1)^{-1/3} + 1/3.$$

Kitaev's lower bound states that $A_{CF}B_{CF} \geq 1/2$. From this, we have

$$A_{OT}f(B_{OT}) \geq A_{CF}B_{CF} \geq 1/2.$$

We now proceed to give the lower bound for the bias. Since f is increasing, we have

$$(\varepsilon + 1/2) \cdot f(\varepsilon + 1/2) \geq A_{OT}f(B_{OT}) \geq A_{CF}B_{CF} \geq 1/2.$$

Solving the inequality we show that ε must satisfy

$$\varepsilon \geq \frac{1}{2}\left(\sqrt{\frac{1}{2} + 2\sqrt{2}} - \sqrt{\frac{1}{2}}\right) - \frac{1}{2} \approx 0.0586.$$

3.2 Proof of the Learning-In-Sequence Lemma

The Learning-in-Sequence Lemma follows from the following simple geometric result.

▶ Proposition 2. Let $|\psi\rangle$ be a pure state and let $\{C, I - C\}$ and $\{D, I - D\}$ be two projective measurements such that

$$\cos^2(\theta) := \|C|\psi\rangle\|_2^2 \geq \frac{1}{2} \quad \text{and} \quad \cos^2(\theta') := \|D|\psi\rangle\|_2^2 \geq \frac{1}{2}.$$

Then we have

$$\|DC|\psi\rangle\|_2^2 \geq \cos^2(\theta)\cos^2(\theta + \theta').$$

Proof. Define the following states

$$|X\rangle := \frac{C|\psi\rangle}{\|C|\psi\rangle\|_2}, \quad |X'\rangle := \frac{(I - C)|\psi\rangle}{\|(I - C)|\psi\rangle\|_2}, \quad |Y\rangle := \frac{D|\psi\rangle}{\|D|\psi\rangle\|_2}, \quad |Y'\rangle := \frac{(I - D)|\psi\rangle}{\|(I - D)|\psi\rangle\|_2}.$$

Then we can write $|\psi\rangle = \cos(\theta)|X\rangle + e^{i\alpha}\sin(\theta)|X'\rangle$ and $|\psi\rangle = \cos(\theta')|Y\rangle + e^{i\beta}\sin(\theta')|Y'\rangle$ with $\alpha, \beta \in \mathbb{R}$. Then we have

$$\|DC|\psi\rangle\|_2^2 = \cos^2(\theta)\,\|D|X\rangle\|_2^2 \geq \cos^2(\theta)|\langle Y|X\rangle|^2 \geq \cos^2(\theta)\cos^2(\theta + \theta').$$

◀

We now prove Lemma 1.

Proof. Let $|\Omega\rangle_{\mathcal{AB}}$ be the joint pure state shared by Alice and Bob, where $\mathcal{A}$ is the space controlled by Alice and $\mathcal{B}$ the space controlled by Bob.

Let $M = \{M_{x_0,x_1}\}_{x_0,x_1\in\{0,1\}}$ be Alice's projective measurement on $\mathcal{A}$ to determine her outputs x_0, x_1. Let $P = \{P_0, P_1\}$ be Bob's projective measurement that allows him to guess x_0 with probability $p = \cos^2(\theta)$ and $Q = \{Q_0, Q_1\}$ be Bob's projective measurement that allows him to guess x_1 with probability $q = \cos^2(\theta')$. These measurements are on $\mathcal{B}$ only. Recall that $a = \frac{p+q}{2} = \frac{\cos^2(\theta)+\cos^2(\theta')}{2}$. We consider the following projections on $\mathcal{AB}$:

$$C = \sum_{x_0,x_1} M_{x_0,x_1} \otimes P_{x_0} \quad \text{and} \quad D = \sum_{x_0,x_1} M_{x_0,x_1} \otimes Q_{x_1}.$$

C (resp. D) is the projection on the subspace where Bob guesses correctly the first bit (resp. the second bit) after applying P (resp. Q).

A strategy for Bob to learn both bits is simple: apply the two measurements P and Q one after the other, where the first one is chosen uniformly at random.

The projection on the subspace where Bob guesses (x_0, x_1) when applying P then Q is

$$E = \sum_{x_0,x_1} M_{x_0,x_1} \otimes Q_{x_1} P_{x_0} = DC.$$

Similarly, the projection on the subspace where Bob guesses (x_0, x_1) when applying Q then P is

$$F = \sum_{x_0,x_1} M_{x_0,x_1} \otimes P_{x_0} Q_{x_1} = CD.$$

With this strategy Bob can guess both bits with probability

$$\begin{aligned}
\frac{1}{2}\left(||E|\Omega\rangle||_2^2 + ||F|\Omega\rangle||_2^2\right) &= \frac{1}{2}\left(||DC|\Omega\rangle||_2^2 + ||CD|\Omega\rangle||_2^2\right) \\
&\geq \frac{1}{2}\left(\cos^2(\theta) + \cos^2(\theta')\right)\cos^2(\theta+\theta') \\
&\geq \frac{1}{2}\left(\cos^2(\theta) + \cos^2(\theta')\right)\left(\cos^2(\theta) + \cos^2(\theta') - 1\right)^2 \\
&= a(2a-1)^2.
\end{aligned}$$

Note that we can use Proposition 2 since Bob's optimal measurement to guess x_0 and x_1 succeeds for each bit with probability at least $1/2$. ◀

4 A Two-Message Protocol With Bias 1/4

We present in this section a random-OT protocol with bias $1/4$. This implies, as we have mentioned, an OT protocol with inputs with the same bias.

Random Oblivious Transfer Protocol

1. Bob chooses $b \in_R \{0,1\}$ and creates the state $|\phi_b\rangle := \frac{1}{\sqrt{2}}|bb\rangle + \frac{1}{\sqrt{2}}|22\rangle$.
2. Alice chooses $x_0, x_1 \in_R \{0,1\}$ and applies the unitary $|a\rangle \to (-1)^{x_a}|a\rangle$, where $x_2 := 0$, to half of Bob's state.
3. Alice returns the qutrit to Bob who now has the state $|\psi_b\rangle := \frac{(-1)^{x_b}}{\sqrt{2}}|bb\rangle + \frac{1}{\sqrt{2}}|22\rangle$.
4. Bob performs on the state $|\psi_b\rangle$ the measurement $\{\Pi_0 = |\phi_b\rangle\langle\phi_b|, \Pi_1 := |\phi_b'\rangle\langle\phi_b'|, I - \Pi_0 - \Pi_1\}$, where $|\phi_b'\rangle := \frac{1}{\sqrt{2}}|bb\rangle - \frac{1}{\sqrt{2}}|22\rangle$.
   If the outcome is Π_0 then $x_b = 0$, if it is Π_1 then $x_b = 1$, otherwise he aborts.

It is clear that Bob can learn x_0 or x_1 perfectly. Moreover, note that if he sends half of the state $\frac{1}{\sqrt{2}}|00\rangle + \frac{1}{\sqrt{2}}|11\rangle$ then he can also learn $x_0 \oplus x_1$ perfectly (although in this case he does not learn either of x_0 or x_1). We now show that it is impossible for him to perfectly learn both x_0 and x_1 and also that his bit is not completely revealed to a cheating Alice.

▶ **Theorem 2.** *In the protocol described above, we have* $A_{OT} = B_{OT} = \frac{3}{4}$.

In the full version, we prove this theorem. In the previous section we have shown that no protocol has bias lower than 0.0586 by showing that $A_{OT}f(B_{OT}) \geq 1/2$. In this section we presented a protocol with bias 0.25 and it can be calculated that for this protocol we have $A_{OT}f(B_{OT}) = \frac{3}{4}f\left(\frac{3}{4}\right) \approx 0.709$. It remains an open problem to determine the bias of an optimal protocol.

5 Oblivious Transfer as a Forcing Primitive

Here we discuss a variant of oblivious transfer, as a generalization of coin flipping, that can be analyzed using an extension of Kitaev's semidefinite programming formalism.

▶ **Definition 3** (Forcing Oblivious Transfer). A k-out-of-n forcing oblivious transfer protocol, denoted here as $\binom{n}{k}$-fOT, with *forcing bias* ε is a protocol satisfying:

- Alice outputs n random bits $x := (x_1, \ldots, x_n)$
- Bob outputs a random index set b of k indices and bit string x_b consisting of x_i for $i \in b$
- $A_{b,x_b} := \sup\{\Pr[\text{Alice can force Bob to output } (b, x_b)]\} = \dfrac{\varepsilon_A}{\binom{n}{k} \cdot 2^k}$
- $B_x := \sup\{\Pr[\text{Bob can force Alice to output } x]\} = \dfrac{\varepsilon_B}{2^n}$
- The forcing bias of the protocol is defined as $\varepsilon = \max\{\varepsilon_A, \varepsilon_B\}$

where the suprema are taken over all strategies of Alice and Bob.

The main difference in this new primitive is the definition of security. We design protocols to protect against a dishonest party being able to *force* a desired value as the output of the other party. In the previous section (and in the literature) oblivious transfer protocols are designed to protect against the dishonest party *learning* the other party's output. Notice, for example, that in coin flipping we can design protocols to protect against a dishonest party forcing a desired outcome, but both players *learn* the coin outcome perfectly.

The primitive we have defined is indeed a generalization of coin flipping since we can cast the problem of coin flipping as a 1-out-of-1 forcing oblivious transfer protocol. Of course, in $\binom{1}{1}$-fOT Alice always knows Bob's index set so the forcing bias is the only interesting notion of security in this case.

We define the bias ε as a multiplicative factor instead of additive since the honest probabilities can be different and in this case our definition makes more sense. To relate this bias to the one previously studied in coin flipping we have that coin flipping protocols with bias $\varepsilon \leq \sqrt{2} + \delta$ exist for any $\delta > 0$, see [CK09], and weak coin flipping protocols with bias $\varepsilon \leq 1 + \delta$ exist for any $\delta > 0$, see [Moc07].

5.1 Extending Kitaev's Lower Bound to Forcing Oblivious Transfer

We now extend Kitaev's formalism from the setting of coin flipping to the more general setting of $\binom{n}{k}$-fOT.

Suppose Alice and Bob have private spaces $\mathcal{A}$ and $\mathcal{B}$, respectively, and both have access to a message space $\mathcal{M}$ each initialized in state $|0\rangle$. Then, we can define an m-round $\binom{n}{k}$-fOT protocol using the following parameters:

- Alice's unitary operators $U_{A,1}, \ldots, U_{A,m}$ which act on $\mathcal{A} \otimes \mathcal{M}$
- Bob's unitary operators $U_{B,1}, \ldots, U_{B,m}$ which act on $\mathcal{M} \otimes \mathcal{B}$
- Alice's POVM $\{\Pi_{A,abort}\} \cup \{\Pi_{A,x} : x \in \mathbb{Z}_2^n\}$ acting on $\mathcal{A}$, one for each outcome
- Bob's POVM $\{\Pi_{B,abort}\} \cup \left\{\Pi_{B,(b,x_b)} : b \text{ a k-element subset of n indices}, x_b \in \mathbb{Z}_2^k\right\}$ acting on $\mathcal{B}$, one for each outcome.

We now show the criteria for which the parameters above yield a proper $\binom{n}{k}$-fOT protocol. In a proper protocol we require that Alice and Bob's measurements are consistent and that the outcomes are uniformly random when the protocol is followed honestly. Define

$$|\psi\rangle := (I_{\mathcal{A}} \otimes U_{B,m})(U_{A,m} \otimes I_{\mathcal{B}}) \cdots (I_{\mathcal{A}} \otimes U_{B,1})(U_{A,1} \otimes I_{\mathcal{B}})|0\rangle_{\mathcal{A}\otimes\mathcal{M}\otimes\mathcal{B}}$$

to be the state at the end of an honest run of the protocol. Then, we require the unitary and measurement operators to satisfy the following condition:

$$\left\|(\Pi_{A,x} \otimes I_{\mathcal{M}} \otimes \Pi_{B,(b,x_b)})|\psi\rangle\right\|_2^2 = \frac{1}{\binom{n}{k}2^n} \text{ for } (x, b, x_b) \text{ consistent.}$$

Similar to coin flipping, we can capture cheating strategies as semidefinite programs. Bob can force Alice to output a specific $x \in \mathbb{Z}_2^n$ with maximum probability equal to the optimal value of the following semidefinite program

$$\begin{array}{lrcll} B_x = \max & \langle \Pi_{A,x} \otimes I_{\mathcal{M}}, \rho_{A,N} \rangle & & & \\ \text{subject to} & \mathrm{Tr}_{\mathcal{M}}(\rho_{A,0}) & = & |0\rangle\langle 0|_{\mathcal{A}} & \\ & \mathrm{Tr}_{\mathcal{M}}(\rho_{A,j}) & = & \mathrm{Tr}_{\mathcal{M}}(U_{A,j}\rho_{A,j-1}U_{A,j}^*), & \text{for } j \in \{1,\ldots,N\} \\ & \rho_{A,0},\ldots,\rho_{A,N} & \in & \mathrm{Pos}(\mathcal{A}\otimes\mathcal{M}), & \text{for } j \in \{0,\ldots,N\} \end{array}$$

where $\mathrm{Pos}(\mathcal{H})$ is the set of positive semidefinite matrices over the Hilbert space $\mathcal{H}$. The states ρ_i represent the part of the state under Alice's control after Bob sends his i'th message. The constraints above are necessary since Bob cannot apply a unitary on $\mathcal{A}$. They are also sufficient since Bob can maintain a purification during the protocol consistent with the states above to achieve a cheating probability given by the corresponding objective value.

To capture Alice's cheating strategies we can do the same as for cheating Bob and examine the states under Bob's control during the course of the protocol. That is, Alice can force Bob to output a specific k-element subset b and $x_b \in \mathbb{Z}_2^k$ with maximum probability equal to the optimal value of the following semidefinite program

$$\begin{array}{lrcll} A_{b,x_b} = \max & \langle I_{\mathcal{M}} \otimes \Pi_{B,(b,x_b)}, \rho_{B,N} \rangle & & & \\ \text{subject to} & \mathrm{Tr}_{\mathcal{M}}(\rho_{B,0}) & = & |0\rangle\langle 0|_{\mathcal{B}} & \\ & \mathrm{Tr}_{\mathcal{M}}(\rho_{B,j}) & = & \mathrm{Tr}_{\mathcal{M}}(U_{B,j}\rho_{B,j-1}U_{B,j}^*), & \text{for } j \in \{1,\ldots,N\} \\ & \rho_{B,0},\ldots,\rho_{B,N} & \in & \mathrm{Pos}(\mathcal{M}\otimes\mathcal{B}), & \text{for } j \in \{0,\ldots,N\} \end{array}$$

The proofs that these capture the optimal cheating probabilities are the same as those for coin flipping in [Kit03] and [ABDR04]. Using these semidefinite programs we can prove the following theorem.

▶ **Theorem 3.** *In any* $\binom{n}{k}$-fOT *protocol and consistent* b, x, x_b *we have*

$$B_x \cdot A_{b,x_b} \geq \Pr[\textit{Alice honestly outputs } x \textit{ and Bob honestly outputs } (b, x_b)] = \frac{1}{\binom{n}{k}2^n}.$$

In particular, the forcing bias satisfies $\varepsilon \geq \sqrt{2}^k$.

Once we extended the semidefinite programming formulation, the proof of the theorem follows almost directly from the proof in [Kit03] and [ABDR04] for coin flipping except that the honest outcome probabilities are different in our case. Namely, for $|\psi\rangle$ defined above, we have

$$\left\|(\Pi_{A,x} \otimes I_{\mathcal{M}} \otimes \Pi_{B,(b,x_b)})|\psi\rangle\right\|_2^2 = \frac{1}{\binom{n}{k}2^n}$$

when x, b, and x_b are consistent and 0 otherwise.

5.2 A Protocol with Optimal Forcing Bias

In this section we prove Theorem 4. First, consider the following protocol which achieves the bound in Theorem 3 but is asymmetric. Alice sends n random bits to Bob. Bob then outputs b, a random k-index subset of n indices, and x_b. In this protocol Bob can force a desired outcome with probability $\frac{1}{2^n}$ and Alice can force a desired outcome with probability $\frac{1}{\binom{n}{k}}$. Thus the product of the cheating probabilities is optimal, that is it achieves the lower bound in Theorem 3. However the protocol is asymmetric. This can be easily remedied using coin flipping. We present an optimal protocol with this security definition.

An Optimal $\binom{n}{k}$-fOT Protocol with Forcing Bias $\sqrt{2}^k$

1. Bob outputs a random index set b of k indices and sends the result to Alice.
2. Alice and Bob play a coin flipping game with bias $\sqrt{2}+\delta$ (for a $\delta > 0$ sufficiently small) to determine each bit in x_b.
3. Alice randomly chooses her bits not in b.

▶ **Theorem 4.** *For any $\gamma > 0$ we can choose a $\delta > 0$ such that the $\binom{n}{k}$-fOT protocol above satisfies*

$$A_{b,x_b} \leq \frac{\sqrt{2}^k(1+\gamma)}{\binom{n}{k}\cdot 2^k} \quad \text{and} \quad B_x \leq \frac{\sqrt{2}^k(1+\gamma)}{2^n}.$$

We prove this theorem in the final version. Note that we have coin flipping protocols with $poly(m)$ rounds that achieve $\delta = \frac{1}{poly(m)}$. Hence, our protocol also achieves $\gamma = \frac{1}{poly(m)}$ with $poly(m)$ rounds.

References

ABDR04 Andris Ambainis, Harry Buhrman, Yevgeniy Dodis, and Hein Rohrig. Multiparty quantum coin flipping. In *CCC '04: Proceedings of the 19th IEEE Annual Conference on Computational Complexity*, pages 250–259, Washington, DC, USA, 2004. IEEE Computer Society.

Amb01 Andris Ambainis. A new protocol and lower bounds for quantum coin flipping. In *STOC '01: Proceedings of the thirtieth annual ACM symposium on Theory of computing*, Washington, DC, USA, 2001. IEEE Computer Society.

Amb02 Andris Ambainis. Lower bound for a class of weak quantum coin flipping protocols, 2002. quant-ph/0204063.

ATVY00 Dorit Aharonov, Amnon Ta-Shma, Umesh V. Vazirani, and Andrew C. Yao. Quantum bit escrow. In *STOC '00: Proceedings of the thirty-second annual ACM symposium on Theory of computing*, pages 705–714, New York, NY, USA, 2000. ACM.

BB84 Bennett and Brassard. Quantum cryptography: Public key distribution and coin tossing. in Proc. Of IEEE Inter. Conf. on Computer Systems and Signal Processing, Bangalore, Kartarna, (Institute of Electrical and Electronics Engineers, New York, 1984.

BF10 Niek Bouman and Serge Fehr. Sampling in a quantum population, and applications. In *CRYPTO 2010*, 2010.

Blu81 Manuel Blum. Coin flipping by telephone. In *CRYPTO*, pages 11–15, 1981.

CK09 André Chailloux and Iordanis Kerenidis. Optimal quantum strong coin flipping. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:527–533, 2009.

Cré87 Claude Crépeau. Equivalence between two flavours of oblivious transfer. In *Advances in Cryptology: CRYPTO '87*, 1987.

DKSW07 Giacomo Mauro D'Ariano, Dennis Kretschmann, Dirk Schlingemann, and Reinhard F. Werner. Reexamination of quantum bit commitment: the possible and the impossible. *Physical Review A*, 76:032328, 2007.

DW09 Andrew Drucker and Ronald de Wolf. Quantum proofs for classical theorems, 2009. quant-ph/0910.3376.

EGL82 Shimon Even, Oded Goldreich, and Abraham Lempel. A randomized protocol for signing contracts. In *Advances in Cryptology: Proceedings of CRYPTO 82*, 1982.

FS09 Serge Fehr and Christian Schaffner. Composing quantum protocols in a classical environment. In *Theory of Cryptography—TCC '09*, volume 5444 of *Lecture Notes in Computer Science*, pages 350–367. Springer-Verlag, 2009.

JRS02 Rahul Jain, Jaikumar Radhakrishnan, and Pranab Sen. A theorem about relative entropy of quantum states with an application to privacy in quantum communication. In *Proceedings of 43rd IEEE Symposium on Foundations of Computer Science (FOCS)*, 2002.

Kil88 Joe Kilian. Founding cryptography on oblivious transfer. In *STOC '88: Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 20 – 31, New York, NY, USA, 1988. ACM Press.

Kit03 A Kitaev. Quantum coin-flipping. Presentation at the 6th workshop on quantum information processing (qip 2003), 2003.

KN04 I. Kerenidis and A. Nayak. Weak coin flipping with small bias. *Inf. Process. Lett.*, 89(3):131–135, 2004.

LC97 Hoi-Kwong Lo and H. F. Chau. Is quantum bit commitment really possible? *Phys. Rev. Lett.*, 78(17):3410–3413, Apr 1997.

Lo97 Hoi-Kwong Lo. Insecurity of quantum secure computations. *Phys. Rev. A*, 56(2):1154–1162, 1997.

May97 Dominic Mayers. Unconditionally secure quantum bit commitment is impossible. *Phys. Rev. Lett.*, 78(17):3414–3417, Apr 1997.

Moc05 C. Mochon. Large family of quantum weak coin-flipping protocols. *Phys. Rev. A*, 72(2):022341–+, August 2005.

Moc07 Carlos Mochon. Quantum weak coin flipping with arbitrarily small bias. WCF, 2007. quant-ph:0711.4114.

Nay99 Ashwin Nayak. Optimal lower bounds for quantum automata and random access codes. *Foundations of Computer Science, Annual IEEE Symposium on*, 0:369, 1999.

NC00 Michael A. Nielsen and Isaac L. Chuang. *Quantum computation and quantum information*. Cambridge University Press, New York, NY, USA, 2000.

NS03 Ashwin Nayak and Peter Shor. Bit-commitment-based quantum coin flipping. *Phys. Rev. A*, 67(1):012304, Jan 2003.

Rab81 Michael Rabin. How to exchange secrets by oblivious transfer. In *Technical Report TR-81, Aiken Computation Laboratory, Harvard University*, 1981.

SR01 R. W. Spekkens and T. Rudolph. Degrees of concealment and bindingness in quantum bit commitment protocols. *Physical Review A*, 65:012310, 2001.

SR02 Robert Spekkens and Terry Rudolph. Quantum protocol for cheat-sensitive weak coin flipping. *Phys. Rev. Lett.*, 89(22):1–4, Nov 2002.

SSS09 Louis Salvail, Christian Schaffner, and Miroslava Sotakova. On the power of two-party quantum cryptography. In *ASIACRYPT 2009*, 2009.

Yao95 Andrew Yao. Security of quantum protocols against coherent measurements. In *Proceedings of 26th Annual ACM Symposium on the Theory of Computing*, pages 67–75, 1995.

Minimizing Busy Time in Multiple Machine Real-time Scheduling

Rohit Khandekar[1], Baruch Schieber[1], Hadas Shachnai[2], and Tami Tamir[3]

1 **IBM T.J. Watson Research Center**
{rohitk,sbar}@us.ibm.com
2 **Computer Science Department, Technion**
hadas@cs.technion.ac.il
3 **School of Computer Science, The Interdisciplinary Center**
tami@idc.ac.il

Abstract

We consider the following fundamental scheduling problem. The input consists of n jobs to be scheduled on a set of machines of bounded capacities. Each job is associated with a release time, a due date, a processing time and demand for machine capacity. The goal is to schedule all of the jobs non-preemptively in their release-time-deadline windows, subject to machine capacity constraints, such that the total busy time of the machines is minimized. Our problem has important applications in power-aware scheduling, optical network design and unit commitment in power systems. Scheduling to minimize busy times is APX-hard already in the special case where all jobs have the same (unit) processing times and can be scheduled in a fixed time interval.

Our main result is a 5-approximation algorithm for general instances. We extend this result to obtain an algorithm with the same approximation ratio for the problem of scheduling *moldable* jobs, that requires also to determine, for each job, one of several processing-time vs. demand configurations. Better bounds and exact algorithms are derived for several special cases, including proper interval graphs, intervals forming a clique and laminar families of intervals.

Keywords and phrases real-time scheduling, busy time, preemption, approximation algorithm

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.169

1 Introduction

Traditional research interest in cluster systems has been high performance, such as high throughput, low response time, or load balancing. In this paper we focus on minimizing machine busy times, a fundamental problem in cluster computing, which aims at reducing power consumption (see, e.g., [22] and the references therein).

Given is a set of n jobs $\mathcal{J} = \{J_1, \ldots, J_n\}$ that need to be scheduled on a set of identical machines, each of which having *capacity* g, for some $g \geq 1$. Each job J has a release time $r(J)$, a due date $d(J)$, a processing time (or, length) $p(J) > 0$ (such that $d(J) \geq r(J) + p(J)$) and a *demand* $1 \leq R(J) \leq g$ for machine capacity; this is the amount of capacity required for processing J on any machine.

A feasible solution schedules each job J on a machine M *non-preemptively* during a time interval $[t(J), t(J) + p(J))$, such that $t(J) \geq r(J)$ and $t(J) + p(J) \leq d(J)$, and the total demand of jobs running at any given time on each machine is at most g. We say that a machine M is *busy* at time t if there is at least one job J scheduled on M such that $t \in [t(J), t(J) + p(J))$; otherwise, M is *idle* at time t. We call the time period in which a machine M is busy its *busy period* and denote its length by $\texttt{busy}(M)$. The goal is to find a

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 169–180

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

feasible schedule of all jobs on a set of machines such that the total busy time of the machines, given by $\sum_M \texttt{busy}(M)$, is minimized. We consider the offline version of this problem where the entire input is given in advance.

Note that the number of machines to be used is part of the output (and can take any integral value $m \geq 1$). Indeed, a solution which minimizes the total busy time may not be optimal in the number of machines used. Also, it is NP-hard to approximate our problem within ratio better than $\frac{3}{2}$, already in the special case where all jobs have the same (unit) processing times and can be scheduled in a fixed time interval, by a simple reduction from the subset sum problem.[1]

1.1 Applications

Power-aware scheduling. The objective of power-aware scheduling is to minimize the power consumption for running a cluster of machines, while supporting Service Level Agreements (SLAs). SLAs, which define the negotiated agreements between service providers and consumers, include quality of service parameters such as demand for a computing resource and a deadline. The power consumption of a machine is assumed to be proportional to the time the machine is in *on* state. While *on*, a machine can process several tasks simultaneously. The number of these tasks hardly affects the power consumption, but must be below the given machine's capacity. Thus, we get an instance of our problem of minimizing the total busy time of the schedule.

Optical network design. Communication in an optical network is achieved by *lightpaths*, which are simple paths in the network. Hardware cost for operating such a network is proportional to the number of switching units such as Optical Add-Drop Multiplexers (or OADMs) installed at the nodes in the network. A lightpath j with transmission rate $R(j)$ uses that capacity in an OADM at each internal node on the path. Assuming that the OADMs have transmission capacity g, one would like to "groom" multiple lightpaths together so that their aggregate transmission rate is at most g. It is easy to see that this grooming problem [11, 10, 9] for minimizing switching costs in optical networks with *path-topologies* can be reduced to our real-time scheduling problem.

Unit commitment given future demand. The Unit commitment in power systems involves determining the start-up and shut-down schedule of generation units to meet the required demand. This is one of the major problems in power generation (see, e.g., [24, 2] and the references therein). Under-commitment of units would result in extra cost due to the need to purchase the missing power in the spot market, while overcommitment would result in extra operating cost. In a simplified version of the problem, assume that all generation units have the same capacity and that the blocks of future demands are given in advance. This yields an instance of our real-time scheduling problem, where each generation unit corresponds to a machine, and each block of demand corresponds to a job.

1.2 Related Work

Job scheduling on parallel machines has been widely studied (see, e.g., the surveys in [7, 4]). In particular, much attention was given to *interval scheduling* [14], where jobs are given

[1] Given the integers $a_1, \ldots, a_n \in \{1, \ldots, g\}$ summing to $2g$, the subset sum problem (SSP) is to determine if there is a subset of numbers adding to exactly g. In the reduction, we create for each i, a job J_i with demand a_i, release time 0, processing time 1 and deadline 1. The *yes* instance of SSP results in the busy time of 2 while the *no* instance results in the busy time of 3.

as intervals on the real line, each representing the time interval in which a job should be processed. Each job has to be processed on some machine, and it is commonly assumed that a machine can process a *single* job at any time. Some of the earlier work on interval scheduling considers the problem of scheduling a feasible subset of jobs whose total weight is maximized, i.e., a *maximum weight independent set* (see, e.g., [1] and the survey in [13]).

There is wide literature also on *real-time scheduling*, where each job has to be processed on some machine during a time interval between its release time and due date. There are studies of real-time scheduling with demands, where each machine has some capacity; however, to the best of our knowledge, all of this prior art refers to objectives other than minimizing the total busy time of the schedule (see, e.g., [1, 18, 5, 6]). There has been earlier work also on the problem of scheduling the jobs on a set of machines so as to minimize the total cost (see, e.g., [3]), but in these works the cost of scheduling each job is *fixed*. In our problem, the cost of scheduling each of the jobs depends on the other jobs scheduled on the same machine in the corresponding time interval; thus, it may change over time and among different machines. Scheduling *moldable* jobs, where each job can have varying processing times, depending on the amount of resources allotted to this job, has been studied using classic measures, such as minimum makespan, or minimum (weighted) sum of completion times (see, e.g., [21, 16] and a comprehensive survey in [20]). Scheduling moldable jobs differs from *malleable* jobs in which the amount of resources allotted to jobs may change over their execution [15].

Our study relates also to *batch scheduling* of conflicting jobs, where the conflicts are given as an interval graph. In the *p-batch* scheduling model (see e.g. Chapter 8 in [4]), a set of jobs can be processed jointly. All the jobs in the batch start simultaneously, and the completion time of a batch is the last completion time of any job in the batch. (For known results on batch scheduling, see e.g., [4, 19].) Our scheduling problem differs from batch scheduling in several aspects. While each machine can process (at most) g jobs simultaneously, for some $g \geq 1$, the jobs need not be partitioned to batches, i.e., each job can start at different time. Also, while in known batch scheduling problems the set of machines is given, we assume that *any* number of machines can be used for the solution. Finally, while common measures in batch scheduling refer to the maximum completion time of a batch, or a function of the completion times of the jobs, we consider the total busy times of the machines. Other work on energy minimization consider utilization of machines with variable capacities, corresponding to their voltage consumption [17], and scheduling of jobs with precedence constraints [12, 25].

The complexity of our scheduling problem was studied in [23]. This paper shows that the problem is NP-hard already for $g = 2$, where the jobs are intervals on the line. Flammini et al. [9] consider our scheduling problem where jobs are given as intervals on the line with unit demand. For this version, they give a 4-approximation algorithm for general inputs and better bounds for some subclasses of inputs. In particular, the paper presents a 2-approximation algorithm for instances where no interval is properly contained in another (i.e., the input forms a *proper* interval graph), and a $(2 + \varepsilon)$-approximation for bounded lengths instances, i.e., the length (or, processing time) of any job is bounded by some fixed integer d.[2] A 2-approximation algorithm was given in [9] for instances where any two intervals intersect, i.e., the input forms a clique (see also in [10]). In this paper we improve and extend the results of [9].

[2] A slight modification of the algorithm yields an improved bound of $1 + \varepsilon$, where $\varepsilon > 0$ is an input parameter.

1.3 Our Results

Our main result is a 5-approximation algorithm for real-time scheduling of *moldable* jobs. Before summarizing our results, we introduce some notation. Denote by $I(J)$ the interval $[r(J), d(J))$ in which J can be processed.

▶ **Definition 1.** An instance $\mathcal{J}$ is said to have *interval jobs* if $d(J) = r(J) + p(J)$ holds for all jobs $J \in \mathcal{J}$. An instance $\mathcal{J}$ with interval jobs is called

- *proper* if for any two jobs $J, J' \in \mathcal{J}$, neither $I(J) \subseteq I(J')$ nor $I(J') \subseteq I(J)$ holds.
- *laminar* if the intervals $I(J)$ for all jobs J form a laminar family, i.e., for any two jobs $J, J' \in \mathcal{J}$, we have $I(J) \cap I(J') = \emptyset$ or $I(J) \subseteq I(J')$, or $I(J') \subseteq I(J)$.
- a *clique* if intervals $I(J)$ for all jobs J form a clique, i.e., for any two jobs $J, J' \in \mathcal{J}$, we have $I(J) \cap I(J') \neq \emptyset$.

We first prove the following result for instances with interval jobs.

▶ **Theorem 2.** *There exists a* 5*-approximation algorithm for real-time scheduling instances with interval jobs. Furthermore, if the instance is proper, there exists a* 2*-approximation algorithm.*

We use the above algorithm, as a subroutine, to design our algorithm for the general real-time scheduling problem.

▶ **Theorem 3.** *There exists a* 5*-approximation algorithm for the real-time scheduling problem.*

Next, we consider an extension to real-time scheduling of *moldable* jobs. In this generalization, a job does not have a fixed processing time and demand; rather, it can be scheduled in one of several possible *configurations*. More precisely, for each job $J \in \mathcal{J}$, we have $q \geq 1$ configurations, where configuration i is given by a pair $(p_i(J), R_i(J))$. The problem involves deciding which configuration i_J is to be used in the schedule for each job J. Once a configuration i_J is finalized for a job J, its processing time and demand are given by $p_{i_J}(J)$ and $R_{i_J}(J)$, respectively. We assume that q is polynomially bounded in the input size.

▶ **Theorem 4.** *There exists a* 5*-approximation algorithm for real-time scheduling of* moldable *jobs.*

Finally, we present improved bounds for some cases of instances with interval jobs.

▶ **Theorem 5.** *Consider an instance $\mathcal{J}$ consisting of interval jobs with* unit *demands. There exist* (*i*) *a polynomial time exact algorithm if $\mathcal{J}$ is laminar, and* (*ii*) *a PTAS if $\mathcal{J}$ is a clique.*

1.4 Preliminaries

▶ **Definition 6.** Given a time interval $I = [s, t)$, the length of I is $\texttt{len}(I) = t - s$. This extends to a set $\mathcal{I}$ of intervals; namely, the length of $\mathcal{I}$ is $\texttt{len}(\mathcal{I}) = \sum_{I \in \mathcal{I}} \texttt{len}(I)$. We define the span of $\mathcal{I}$ as $\texttt{span}(\mathcal{I}) = \texttt{len}(\cup \mathcal{I})$.

Note that $\texttt{span}(\mathcal{I}) \leq \texttt{len}(\mathcal{I})$ and equality holds if and only if $\mathcal{I}$ is a set of pairwise disjoint intervals.

Given an instance $\mathcal{J}$ and machine capacity $g \geq 1$, we denote by $\textsc{opt}(\mathcal{J})$ the cost of an optimal solution, that is, a feasible schedule in which the total busy time of the machines is minimized. Also, we denote by $\textsc{opt}_\infty(\mathcal{J})$ the cost of the optimum solution for the instance $\mathcal{J}$, assuming that the capacity is $g = \infty$. For any job J, let $w(J) = R(J) \cdot p(J)$ denote the total work required by job J, then for a set of jobs $\mathcal{J}$, $w(\mathcal{J}) = \sum_{J \in \mathcal{J}} w(J)$ is the total work required by the jobs in $\mathcal{J}$. The next observation gives two immediate lower bounds for the cost of any solution.

▶ **Observation 7.** *For any instance $\mathcal{J}$ and machine capacity $g \geq 1$, the following bounds hold.*

- *The work bound:* $\text{OPT}(\mathcal{J}) \geq \frac{w(\mathcal{J})}{g}$.
- *The span bound:* $\text{OPT}(\mathcal{J}) \geq \text{OPT}_\infty(\mathcal{J})$.

The work bound holds since g is the maximum capacity that can be allocated by a single machine at any time. The span bound holds since the busy-time does not increase by relaxing the capacity constraint.

While analyzing any schedule S that is clear from the context, we number the machines as $M_1, M_2, \ldots$, and denote by $\mathcal{J}_i$ the set of jobs assigned to machine M_i under the schedule S. W.l.o.g., the busy period of a machine M_i is contiguous; otherwise, we can divide the busy period to contiguous intervals and assign the jobs of each contiguous interval to a different machine. Obviously, this will not change the total busy time. Therefore, we say that a machine M_i has a *busy interval* which starts at the minimum start time of any job scheduled on M_i and ends at the maximum completion time of any of these jobs. It follows that the cost of M_i is the length of its busy interval, i.e., $\texttt{busy}(M_i) = \texttt{span}(\mathcal{J}_i)$ for all $i \geq 1$.[3]

2 Interval Scheduling: Theorem 2

2.1 General Instances with Interval Jobs

In this section we present an algorithm for instances with interval jobs, where each job $J \in \mathcal{J}$ may have an arbitrary processing time and any demand $1 \leq R(J) \leq g$. Algorithm *First_Fit_with_Demands* ($FF_{\mathcal{D}}$), shown in the frame below, divides the jobs into two groups, NARROW and WIDE, as given below. It schedules NARROW and WIDE jobs on distinct sets of machines. The WIDE jobs are scheduled arbitrarily, while NARROW jobs are scheduled greedily by considering them one after the other, from longest to shortest. Each job is scheduled on the first machine it can fit. Let $\alpha \in [0, 1]$ be a parameter to be fixed later.

▶ **Definition 8.** For a subset $\mathcal{J}' \subseteq \mathcal{J}$ of jobs, let $\text{NARROW}(\mathcal{J}') = \{J \in \mathcal{J}' \mid R(J) \leq \alpha \cdot g\}$ and $\text{WIDE}(\mathcal{J}') = \{J \in \mathcal{J}' \mid R(J) > \alpha \cdot g\}$.

Algorithm ($FF_{\mathcal{D}}$):

1. Schedule jobs in $\text{WIDE}(\mathcal{J})$ arbitrarily on some machines. Do not use these machines for scheduling any other jobs.
2. Sort the jobs in $\text{NARROW}(\mathcal{J})$ by non-increasing lengths, i.e., $p(J_1) \geq \ldots \geq p(J_{n'})$.
3. For $j = 1, \ldots, n'$ do:
   - **a.** Let m denote the number of machines used for jobs $\{J_1, \ldots, J_{j-1}\}$.
   - **b.** Assign J_j to the first machine that can process it, i.e., find the minimum value of $i : 1 \leq i \leq m$ such that, at any time $t \in I(J_j)$, the total capacity allocated on M_i is at most $g - R(J_j)$.
   - **c.** If no such machine exists open $(m+1)$th machine and schedule J_j on it.

Let $FF_{\mathcal{D}}(\mathcal{J})$ denote the total busy time of the schedule computed by $FF_{\mathcal{D}}$ on job $\mathcal{J}$.

▶ **Theorem 9.** *If $\alpha = 1/4$, for any instance $\mathcal{J}$ with interval jobs, we have*

$$FF_{\mathcal{D}}(\mathcal{J}) \leq \text{OPT}_\infty(\mathcal{J}) + 4 \cdot \frac{w(\mathcal{J})}{g} \leq 5 \cdot \text{OPT}(\mathcal{J}).$$

[3] By $\texttt{span}(\mathcal{J}_i)$ we refer to the span of the set of intervals representing the jobs, as scheduled on M_i.

To prove the theorem we bound the costs of the WIDE and NARROW jobs separately. It is easy to bound the contribution of WIDE jobs to the overall cost. The following lemma follows directly from the definition of WIDE jobs.

▶ **Lemma 10.** *The cost incurred by jobs in* $\text{WIDE}(\mathcal{J})$ *is at most* $\sum_{J\in\text{WIDE}(\mathcal{J})} p(J) \leq \frac{w(\text{WIDE}(\mathcal{J}))}{\alpha\cdot g}$.

The rest of the section is devoted to bounding the cost of the NARROW jobs. The next observation follows from the fact that the *first-fit* algorithm $FF_{\mathcal{D}}$ assigns a job J to machine M_i with $i \geq 2$ only when it could not have been assigned to machines M_k with $k < i$, due to capacity constraints.

▶ **Observation 11.** *Let J be a job assigned to machine M_i by $FF_{\mathcal{D}}$, for some $i \geq 2$. For any machine $M_k, (k < i)$, there is at least one time $t_{i,k}(J) \in I(J)$ and a set $s_{i,k}(J)$ of jobs assigned to M_k such that, for every $J' \in s_{i,k}(J)$, (a) $t_{i,k}(J) \in I(J')$, and (b) $p(J') \geq p(J)$. In addition, $R(J) + \sum_{J'\in s_{i,k}(J)} R(J') > g$.*

In the subsequent analysis, we assume that each job $J \in \mathcal{J}_i$, for $i \geq 2$, fixes a unique time $t_{i,i-1}(J)$ and a unique set of jobs $s_{i,i-1}(J) \subseteq \mathcal{J}_{i-1}$. We say that J *blames* jobs in $s_{i,i-1}(J)$.

▶ **Lemma 12.** *For any $1 \leq i \leq m-1$, we have $g \cdot \texttt{span}(\mathcal{J}_{i+1}) \leq \frac{3\cdot w(\mathcal{J}_i)}{1-\alpha}$.*

Proof. Following Observation 11, for a job $J \in \mathcal{J}_i$, denote by $b(J)$ the set of jobs in $\mathcal{J}_{i+1}$ which blame J, i.e., $b(J) = \{J' \in \mathcal{J}_{i+1} \mid J \in s_{i+1,i}(J')\}$. Let J_L (resp. J_R) be the job with earliest start time (resp. latest completion time) in $b(J)$. Since each job in $b(J)$ intersects J, we have $\texttt{span}(b(J)) \leq p(J) + p(J_L) + p(J_R) \leq 3 \cdot p(J) = 3 \cdot \frac{w(J)}{R(J)}$. Thus,

$$\sum_{J\in\mathcal{J}_i} R(J)\texttt{span}(b(J)) \leq 3 \cdot w(\mathcal{J}_i). \tag{1}$$

Now, we observe that

$$\sum_{J\in\mathcal{J}_i} R(J)\texttt{span}(b(J)) = \int_{t\in\texttt{span}(\mathcal{J}_{i+1})} \sum_{J\in\mathcal{J}_i : t\in\texttt{span}(b(J))} R(J)dt.$$

We bound the right-hand-side as follows. For any $t \in \texttt{span}(\mathcal{J}_{i+1})$, there exists a job $J' \in \mathcal{J}_{i+1}$ with $t \in [r(J'), d(J'))$. For all jobs $J \in s_{i+1,i}(J')$, since $J' \in b(J)$, we have $t \in \texttt{span}(b(J))$. Hence, $\sum_{J\in\mathcal{J}_i : t\in\texttt{span}(b(J))} R(J) \geq \sum_{J\in s_{i+1,i}(J')} R(J)$. By Observation 11 $\sum_{J\in s_{i+1,i}(J')} R(J) > g - R(J') \geq (1-\alpha)\cdot g$. We thus conclude

$$\sum_{J\in\mathcal{J}_i} R(J)\texttt{span}(b(J)) > \texttt{span}(\mathcal{J}_{i+1}) \cdot (1-\alpha) \cdot g. \tag{2}$$

From (1) and (2) we get the lemma. ◀

Proof of Theorem 9: The overall cost of the schedule computed by $FF_{\mathcal{D}}$ is the contribution of WIDE jobs and NARROW jobs. Note that the busy time of M_i, for $1 \leq i \leq m$ is exactly $\texttt{busy}(M_i) = \texttt{span}(\mathcal{J}_i)$. Now from Lemmas 10 and 12, we have that the total cost of $FF_{\mathcal{D}}$ is at most

$$\begin{aligned}
\frac{w(\text{WIDE}(\mathcal{J}))}{\alpha\cdot g} + \sum_{i=1}^{m} \texttt{span}(\mathcal{J}_i) &\leq \frac{w(\text{WIDE}(\mathcal{J}))}{\alpha\cdot g} + \texttt{span}(\mathcal{J}_1) + \sum_{i=1}^{m-1} \frac{3\cdot w(\mathcal{J}_i)}{(1-\alpha)\cdot g} \\
&\leq \frac{w(\text{WIDE}(\mathcal{J}))}{\alpha\cdot g} + \text{OPT}_\infty(\mathcal{J}) + \frac{3\cdot w(\text{NARROW}(\mathcal{J}))}{(1-\alpha)\cdot g} \\
&\leq \text{OPT}_\infty(\mathcal{J}) + \max\left\{\frac{1}{\alpha}, \frac{3}{1-\alpha}\right\} \cdot \frac{w(\mathcal{J})}{g} \\
&\leq \text{OPT}_\infty(\mathcal{J}) + 4 \cdot \frac{w(\mathcal{J})}{g}.
\end{aligned}$$

The second inequality follows since $\texttt{span}(\mathcal{J}_1) \leq \text{OPT}_\infty(\mathcal{J})$. The last inequality holds since $\alpha = 1/4$. The proof now follows from Observation 7.

2.2 Proper Instances with Interval Jobs

We now consider instances in which no job interval is contained in another. The intersection graphs for such instances are known as *proper interval graphs*. The simple greedy algorithm consists of two steps. In the first step, the jobs are sorted by their starting times (note that, in a proper interval graph, this is also the order of the jobs by completion times). In the second step the jobs are assigned to machines greedily in a NextFit manner; that is, each job is added to the currently filled machine, unless its addition is invalid, in which case a new machine is opened.

Greedy Algorithm for Proper Interval Graphs:
1. Sort the jobs in non-decreasing order of release times, i.e., $r(J_1) \leq \ldots \leq r(J_n)$.
2. For $j = 1, \ldots, n$ do: Assign J_j to the currently filled machine if this satisfies the capacity constraint g; otherwise, assign J_j to a new machine and mark it as being current filled.

▶ **Theorem 13.** *Greedy is a 2-approximation algorithm for proper interval graphs.*

Proof. Let D_t be the total demand of jobs active at time t. Also, let M_t^O denote the number of machines active at time t in an optimal schedule, and let M_t^A be the number of machines active at time t in the schedule output by the algorithm. The proofs of the following lemmas are omitted due to lack of space.

▶ **Lemma 14.** *For any t, we have $D_t > g \left\lfloor \frac{M_t^A - 1}{2} \right\rfloor$.*

▶ **Lemma 15.** *For any t, we have $M_t^O \geq M_t^A/2$.*

Therefore, the cost of the output solution is $\int_{t \in \texttt{span}(\mathcal{J})} M_t^A dt \leq \int_{t \in \texttt{span}(\mathcal{J})} 2 \cdot M_t^O dt = 2 \cdot \text{OPT}(\mathcal{J})$, as claimed. ◀

3 Real-time Scheduling: Theorem 3

In this section we show how the results of §2 can extended to scheduling general instances $\mathcal{J}$ where each job J can be processed in the time window $[r(J), d(J))$.

▶ **Lemma 16.** *If there exists a β-approximation algorithm for the real-time scheduling with $g = \infty$, there exists an algorithm that computes a feasible solution to the real-time scheduling problem instance, $\mathcal{J}$, with cost at most $\beta \cdot \text{OPT}_\infty(\mathcal{J}) + 4 \cdot \frac{w(\mathcal{J})}{g}$, thus yielding a $(\beta + 4)$-approximation.*

Proof. We first compute a schedule, called S_∞, with busy-time at most $\beta \cdot \text{OPT}_\infty(\mathcal{J})$, for the given instance with $g = \infty$. Let $[t_\infty(J), t_\infty(J) + p(J)) \subseteq [r(J), d(J))$ be the interval during which job J is scheduled in S_∞. We next create a new instance $\mathcal{J}'$ obtained from $\mathcal{J}$ by replacing $r(J)$ and $d(J)$ with $t_\infty(J)$ and $t_\infty(J) + p(J)$, respectively, for each job J. Note that $\text{OPT}_\infty(\mathcal{J}') \leq \beta \cdot \text{OPT}_\infty(\mathcal{J}) \leq \beta \cdot \text{OPT}(\mathcal{J})$. We then run algorithm $FF_\mathcal{D}$ on instance $\mathcal{J}'$. Theorem 9 implies that the resulting solution has busy-time at most $\text{OPT}_\infty(\mathcal{J}') + 4 \cdot \frac{w(\mathcal{J}')}{g} \leq \beta \cdot \text{OPT}_\infty(\mathcal{J}) + 4 \cdot \frac{w(\mathcal{J})}{g} \leq (\beta + 4) \cdot \text{OPT}(\mathcal{J})$ as claimed. ◀

The following theorem with the above lemma implies a 5-approximation algorithm for the real-time scheduling.

▶ **Theorem 17.** *If $g = \infty$, the real-time scheduling problem is polynomially solvable.*

The rest of this section is devoted to proving the above theorem. To describe our dynamic programming based algorithm, we first identify some useful properties of the optimum schedule. Recall that we can assume, w.l.o.g., that the busy period of each machine is a contiguous interval.

▶ **Lemma 18.** *W.l.o.g., we can assume that the busy period of any machine in the optimum schedule starts at a time given by $d(J) - p(J)$ for some job J and ends at a time given by either $r(J') + p(J')$, for some job J', or $d(J) - p(J) + p(J')$ for some jobs J and J'. Furthermore, we can assume that the start time of any job J is either its release time $r(J)$ or the start time of the busy period of some machine.*

Motivated by Lemma 18, we consider the following definition.

▶ **Definition 19.** A time t is called *interesting* if $t = r(J)$ or $d(J) - p(J)$ for some job J, or $t = r(J) + p(J)$ or $d(J) - p(J) + p(J')$ for some jobs J and J'. Let $\mathcal{T}$ denote the set of interesting times.

Thus, w.l.o.g., we may assume that the busy periods of all machines and placements of all jobs start and end at interesting times. Let the intervals of all the jobs be contained in $[0, T)$. W.l.o.g., we may assume that both 0 and T are interesting times. Note that the number of interesting times is polynomial.

Now we describe our dynamic program. Informally, the algorithm processes the jobs J in the order of non-increasing processing times $p(J)$. It first guesses the placement $[t, t + p(J_1)) \in [r(J_1), d(J_1))$ of job J_1 with largest processing time. Once this is done, the remainder of the problem splits into two *independent* sub-problems: the "left" problem $[0, t)$ and the "right" problem $[t + p(J_1), T)$. This is so because any job J whose interval $[r(J), d(J))$ has an intersection with $[t, t + p(J_1))$ of size at least $p(J)$ can be scheduled inside the interval $[t, t + p(J_1))$ without any extra cost. The "left" sub-problem then estimates the minimum busy time in the interval $[0, t)$ for scheduling jobs whose placement must intersect $[0, t)$; similarly the "right" sub-problem estimates the minimum busy time in the interval $[t+p(J_1), T)$ for scheduling jobs whose placement must intersect $[t+p(J_1), T)$. More formally,

▶ **Definition 20.** Let $t_1, t_2 \in \mathcal{T}$ with $t_2 > t_1$ and $\ell = p(J)$ for some job J. Let $\texttt{jobs}[t_1, t_2, \ell]$ denote the set of jobs in $\mathcal{J}$ whose processing time is at most ℓ and whose placement must intersect the interval $[t_1, t_2)$, i.e.,

$$\texttt{jobs}[t_1, t_2, \ell] = \{J \in \mathcal{J} \mid p(J) \leq \ell, t_1 - r(J) < p(J), d(J) - t_2 < p(J)\}.$$

Let $\texttt{cost}[t_1, t_2, \ell]$ be the minimum busy-time inside the interval $[t_1, t_2)$ for scheduling jobs in $\texttt{jobs}[t_1, t_2, \ell]$ in a feasible manner.

Note that $\texttt{cost}[t_1, t_2, \ell]$ counts the busy-time only inside the interval $[t_1, t_2)$ assuming that the busy-time outside this interval is already "paid for". For convenience, we define $\texttt{jobs}[t_1, t_2, \ell] = \emptyset$ and $\texttt{cost}[t_1, t_2, \ell] = 0$, whenever $t_2 \leq t_1$.

▶ **Lemma 21.** *If $\texttt{jobs}[t_1, t_2, \ell] = \emptyset$ then $\texttt{cost}[t_1, t_2, \ell] = 0$. Otherwise, let $J \in \texttt{jobs}[t_1, t_2, \ell]$ be a job with the longest processing time among the jobs in $\texttt{jobs}[t_1, t_2, \ell]$. Then,*

$$\begin{aligned} \texttt{cost}[t_1, t_2, \ell] \quad = \min_{t \in [r(J), d(J)-p(J)) \cap \mathcal{T}} \quad & \Big(\min\{p(J), t + p(J) - t_1, t_2 - t\} \\ & + \texttt{cost}[t_1, t, p(J)] \qquad + \texttt{cost}[t + p(J), t_2, p(J)] \Big). \end{aligned} \tag{3}$$

Note that the number of interesting times and the number of distinct processing lengths are polynomial. Thus, the quantities $\texttt{cost}[t_1, t_2, \ell]$ for $t_1, t_2 \in \mathcal{T}$ and $\ell = p(J)$ for some $J \in \mathcal{J}$ and their corresponding schedules can be computed, using the relation in Lemma 21, in polynomial time. We finally output the schedule corresponding to $\texttt{cost}[0, T, \max_{J\in\mathcal{J}} p(J)]$. By definition, this gives the optimum solution.

4 Real-time Scheduling for Moldable Jobs: Theorem 4

A job J in an instance $\mathcal{J}$ of the real-time scheduling problem with moldable jobs is described by a release time $r(J)$, a due date $d(J)$, and a set of configurations $\{(p_i(J), R_i(J)\}_{i=1,\ldots,q}$. We assume, w.l.o.g., that $p_i(J) \leq d(J) - r(J)$ for all $1 \leq i \leq q$. The goal is to pick a configuration $1 \leq i_J \leq q$ for each job J and schedule these jobs on machines with a capacity g such that the total busy-time is minimized while satisfying the capacity constraints. Given configurations $\vec{i} = \{i_J\}_{J\in\mathcal{J}}$, let $\mathcal{J}(\vec{i})$ denote the instance of real-time scheduling problem derived from $\mathcal{J}$ by fixing configuration i_J for each job J. Let $\textsc{opt}(\mathcal{J})$ denote the cost of the optimum solution, and let $\vec{i}^* = \{i_J^*\}_{J\in\mathcal{J}}$ denote the configurations used in the optimum schedule. From Observation 7, we have

$$5 \cdot \textsc{opt}(\mathcal{J}) \geq \textsc{opt}_\infty(\mathcal{J}(\vec{i}^*)) + 4 \cdot \frac{w(\mathcal{J}(\vec{i}^*))}{g}. \tag{4}$$

In this section, we prove the following main lemma.

▶ **Lemma 22.** *Given an instance $\mathcal{J}$ of the real-time scheduling with moldable jobs, we can find in polynomial time configurations $\vec{i} = \{i_J\}_{J\in\mathcal{J}}$, such that $\textsc{opt}_\infty(\mathcal{J}(\vec{i})) + 4 \cdot \frac{w(\mathcal{J}(\vec{i}))}{g}$ is minimized.*

Proof. Motivated by Lemma 18 and Definition 19, we define the set of interesting times as follows.

▶ **Definition 23.** A time t is called *interesting* if $t = r(J)$ or $d(J) - p_i(J)$ for some job J and configuration i, or $t = r(J) + p_i(J)$ or $d(J) - p_i(J) + p_{i'}(J')$ for some jobs J and J' and their respective configurations i and i'. Let $\mathcal{T}$ denote the set of interesting times.

Note that the size of $\mathcal{T}$ is polynomial and we can assume, w.l.o.g., that the busy periods of all machines and placements of all jobs start and end at interesting times. Let the intervals of all the jobs be contained in $[0, T)$. W.l.o.g. we can assume that both 0 and T are interesting times. For a job $J \in \mathcal{J}$ and a configuration $1 \leq i_J \leq q$, let $w_{i_J}(J) = p_{i_J}(J) \cdot R_{i_J}(J)$.

▶ **Definition 24.** Let $t_1, t_2 \in \mathcal{T}$ with $t_2 > t_1$ and $\ell = p_i(J)$ for some job J. Let

$$\texttt{jobs}[t_1, t_2, \ell] = \{J \in \mathcal{J} \mid r(J) > t_1 - \ell, d(J) < t_2 + \ell\}.$$

For a choice of configurations $\vec{i} = \{i_J\}_{J\in\texttt{jobs}[t_1,t_2,\ell]}$, let $\texttt{cost}[t_1, t_2, \ell, \vec{i}]$ denote the minimum busy-time inside interval $[t_1, t_2)$ for scheduling jobs in $\texttt{jobs}[t_1, t_2, \ell](\vec{i})$ in a feasible manner. Let $\texttt{ub}[t_1, t_2, \ell]$ be the minimum value of

$$\texttt{cost}[t_1, t_2, \ell, \vec{i}] + 4 \cdot \frac{w(\texttt{jobs}[t_1, t_2, \ell](\vec{i}))}{g},$$

where the minimum is taken over all configurations $\vec{i}$ that satisfy $p_{i_J}(J) \leq \ell$, for all jobs $J \in \texttt{jobs}[t_1, t_2, \ell]$.

As before, $\texttt{cost}[t_1, t_2, \ell, \vec{i}]$ counts the busy-time only inside the interval $[t_1, t_2)$, assuming that the busy-time outside this interval is already "paid for".

▶ **Lemma 25.** *If* $\texttt{jobs}[t_1, t_2, \ell] = \emptyset$ *we have* $\texttt{ub}[t_1, t_2, \ell] = 0$. *If* $t_2 \leq t_1$ *then*

$$\texttt{ub}[t_1, t_2, \ell] = \sum_{J \in \texttt{jobs}[t_1, t_2, \ell]} \min_{i_J : p_{i_J}(J) \leq \ell} 4 \cdot \frac{w_{i_J}(J)}{g}.$$

Otherwise, we have

$$\begin{aligned}
\texttt{ub}[t_1, t_2, \ell] \;=\; & \min_{J \in \texttt{jobs}[t_1, t_2, \ell]} \; \min_{i_J : p_{i_J}(J) \leq \ell} \; \min_{t \in [r(J), d(J) - p_{i_J}(J)) \cap \mathcal{T}} \\
& \Bigg(\min \Big\{ p_{i_J}(J), \max\{0, t + p_{i_J}(J) - t_1\}, \max\{0, t_2 - t\} \Big\} \\
& + \sum_{\substack{J' \in \texttt{jobs}[t_1, t_2, \ell] \setminus \\ (\texttt{jobs}[t_1, \min\{t, t_2\}, p_{i_J}(J)] \cup \texttt{jobs}[\max\{t + p_{i_J}(J), t_1\}, t_2, p_{i_J}(J)])}} \; \min_{i_{J'} : p_{i_{J'}}(J') \leq p_{i_J}(J)} 4 \cdot \frac{w_{i_{J'}}(J')}{g} \\
& + \texttt{ub}[t_1, \min\{t, t_2\}, p_{i_J}(J)] \;+\; \texttt{ub}[\max\{t + p_{i_J}(J), t_1\}, t_2, p_{i_J}(J)] \Bigg). \quad (5)
\end{aligned}$$

Note that the number of interesting times and the number of distinct processing lengths are polynomial. Thus, the quantities $\texttt{ub}[t_1, t_2, \ell]$ for $t_1, t_2 \in \mathcal{T}$ and $\ell = p_i(J)$, for some $J \in \mathcal{J}, 1 \leq i \leq q$ and their corresponding job-configurations and schedules can be computed, using the relation in Lemma 25, in polynomial time. The algorithm finally outputs the job-configurations corresponding to $\texttt{ub}[0, T, \max_{J \in \mathcal{J}, 1 \leq i \leq q} p_i(J)]$. By definition, this proves Lemma 22. ◀

Now, recall that Lemma 16 and Theorem 17 together imply that given an instance $\mathcal{J}(\vec{i})$ of the real-time scheduling problem, we can compute in polynomial time a feasible schedule with busy-time at most $\text{OPT}_\infty(\mathcal{J}(\vec{i})) + 4 \cdot \frac{w(\mathcal{J}(\vec{i}))}{g}$. Thus, equation (4), Lemma 22, Lemma 16, and Theorem 17 together imply that we can find a schedule with cost at most

$$\text{OPT}_\infty(\mathcal{J}(\vec{i})) + 4 \cdot \frac{w(\mathcal{J}(\vec{i}))}{g} \;\leq\; \text{OPT}_\infty(\mathcal{J}(\vec{i}^*)) + 4 \cdot \frac{w(\mathcal{J}(\vec{i}^*))}{g} \;\leq\; 5 \cdot \text{OPT}(\mathcal{J}),$$

thus yielding a 5-approximation.

5 Interval Scheduling with Unit Demands: Theorem 5

In this section, we consider instances with interval jobs, where all jobs have unit demands, i.e., $p(J) = d(J) - r(J)$ and $R(J) = 1$.

5.1 Laminar Instances

We show a polynomial time exact algorithm in case the job intervals $I(J)$, for all jobs J, form a laminar family, i.e., for any two jobs $J, J' \in \mathcal{J}$, it holds that $I(J) \cap I(J') = \emptyset$ or $I(J) \subset I(J')$, or $I(J') \subset I(J)$.

Since the job intervals are laminar, the jobs can be represented by a forest F of rooted trees, where each vertex in a tree $T \in F$ corresponds to a job, and a vertex $v(J)$ is an ancestor of a vertex $v(J')$ if and only if $I(J') \subset I(J)$. Let the level of a vertex be defined as follows. Any root of a tree in the forest is at level 1. For all other vertices v, the level of v is

1 plus the level of its parent. Consider an algorithm which assigns jobs in level ℓ to machine $M_{\lceil \ell/g \rceil}$.

▶ **Theorem 26.** *The algorithm yields an optimal solution for laminar instances.*

Proof. Clearly, the algorithm outputs a feasible solution, since at most g jobs are scheduled on any machine at any time. Let M_t (resp., N_t) be the number of active machines (resp., jobs) at time t. Then $M_t = \lceil N_t/g \rceil$. This proves the claim. ◀

5.2 A PTAS and more for Cliques

In the following we show that if all jobs have unit demands, and the corresponding graph is a clique, then the problem can be approximated within factor $1+\varepsilon$, for any $\varepsilon > 0$. Recall that for general instances of job intervals with unit demands the problem is NP-hard already for $g = 2$ [23]. We show that for inputs that form cliques, the problem with $g = 2$ is solvable in polynomial time.

Since the instance $\mathcal{J}$ forms a clique, there is a time t_0 such that $t_0 \in I(J)$ for all $J \in \mathcal{J}$. The PTAS consists of two main phases. First, it extends the interval lengths, then it finds (using dynamic programming) an optimal schedule of the resulting instance on $m = \lceil n/g \rceil$ machines. Note that the original intervals are sub-intervals of the stretched ones, therefore, any feasible schedule of the stretched intervals induces a feasible schedule.

Approximation Scheme for a Clique:

1. Let $c > 1$ be a constant.
2. Let t_0 be such that $t_0 \in J$ for all $J \in \mathcal{J}$. Let $\texttt{left}(J) = t_0 - r(J)$, $\texttt{right}(J) = d(J) - t_0$. Also, let $\texttt{sh}(J) = \min\{\texttt{left}(J), \texttt{right}(J)\}$ and $\texttt{lo}(J) = \max\{\texttt{left}(J), \texttt{right}(J)\}$ be the length of the short (resp. long) segment of J w.r.t. t_0. If $\texttt{sh}(J)/\texttt{lo}(J) \in ((k-1)/c, k/c]$ for some $1 \le k \le c$, stretch the short segment to round the ratio to k/c.
3. Partition the jobs into $2c-1$ classes. For $\ell \in \{1, \ldots, c\}$, the class ℓ consists of all jobs for which $\texttt{sh}(J)/\texttt{lo}(J) = \ell/c$ and $\texttt{left}(J) \ge \texttt{right}(J)$. For $\ell \in \{c+1, \ldots, 2c-1\}$, the class ℓ consists of all jobs for which $\texttt{sh}(J)/\texttt{lo}(J) = (\ell - c)/c$ and $\texttt{left}(J) < \texttt{right}(J)$. Let n_ℓ be the number of jobs in class ℓ, $1 \le \ell \le 2c-1$.
4. For $i \ge 1$, let $C_i(n'_1, \ldots, n'_{2c-1})$ be the minimum cost of scheduling the longest n'_ℓ jobs of class ℓ, for all ℓ, on i machines. Let $m = \lceil n/g \rceil$. Use dynamic programming to find a schedule achieving $C_m(n_1, \ldots, n_{2c-1})$.

We can show the next result – the proof is omitted due to lack of space.

▶ **Theorem 27.** *For any $\varepsilon \in (0, 1]$, the scheme with $c = 1/\varepsilon$ is a PTAS for any clique.*

The case $g = 2$: In this case we can solve the problem optimally, using a reduction to the minimum-weight perfect matching in a complete graph. We first ensure that the number of jobs is even by adding a dummy job with an empty interval. We next construct a complete graph in which each job corresponds to a vertex, and for every pair i, j, the edge (J_i, J_j) has weight $\texttt{span}(J_i \cup J_j)$. Use Edmond's algorithm [8] to find a minimum-weight perfect matching in the graph. It is easy to see that this matching gives the optimum solution to our problem.

References

1 R. Bar-Yehuda A. Bar-Noy, A. Freund, J. Naor, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *J. of the ACM*, pages 1–23, 2000.

2 L. Belede, A. Jain, and R. Reddy Gaddam. Unit commitment with nature and biologically inspired computing. In *World Congress on Nature and Biologically Inspired Computing (NABIC)*, pages 824–829, 2009.

3 S. Bhatia, J. Chuzhoy, A. Freund, and J. Naor. Algorithmic aspects of bandwidth trading. *ACM Transactions on Algorithms*, 3(1), 2007.

4 P. Brucker. *Scheduling Algorithms, 5th ed.* Springer, 2007.

5 G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *9th Conference on Integer Programming and Combinatorial Optimization (IPCO)*, 2002.

6 B. Chen, R. Hassin, and M. Tzur. Allocation of bandwidth and storage. *IIE Transactions*, 34:501–507, 2002.

7 J. Y-T. Leung (ed.). *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRS Press, 2004.

8 J. Edmonds. Paths, trees and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

9 M. Flammini, G. Monaco, L. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. In *IEEE International Parallel & Distributed Processing Symposium (IPDPS)*, 2009.

10 M. Flammini, G. Monaco, L. Moscardelli, M. Shalom, and S. Zaks. Approximating the traffic grooming problem with respect to adms and oadms. In *14th Euro-Par*, 2008.

11 O. Gerstel, R. Ramaswami, and G. Sasaki. Cost effective traffic grooming in wdm rings. In *INFOCOM*, 1998.

12 J. Kang and S. Ranka. Energy-efficient dynamic scheduling on parallel machines. In *High Performance Computing (HiPC)*, pages 208–219, 2008.

13 M. Y. Kovalyov, C. T. Ng, and T. C. E. Cheng. Fixed interval scheduling: Models, applications, computational complexity and algorithms. *European Journal of Operational Research*, 178(2):331–342, 2007.

14 E. Lawler, J.K. Lenstra, A.H.G.R. Kan, and D. Shmoys. Sequencing and scheduling: Algorithms and complexity. *S. C. Graves, A. H. G. Rinnooy Kan, and P. Zipkin (eds.), Handbooks in Operations Research and Management Science*, 4, 1993.

15 J. Leung, L. Kelly, and J. H. Anderson. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

16 W. T. Ludwig. *Algorithms for Scheduling Malleable and Nonmalleable Parallel Tasks*. PhD thesis, Dept. of Computer Science, Univ. of Wisconsin - Madison, 1995.

17 A. Manzak and C. Chakrabarti. Variable voltage task scheduling algorithms for minimizing energy/power. *IEEE Trans. VLSI Syst. 11(2)*, pages 501–507, 2003.

18 C.A. Phillips, R.N. Uma, and J. Wein. Off-line admission control for general scheduling problems. *J. of Scheduling*, 3:365–381, 2000.

19 M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer, 2008.

20 U.M. Schwarz. Tightness results for malleable task scheduling algorithms. In *Parallel Processing and Applied Mathematics*, 2009.

21 J. Turek, J.L. Wolf, and P. S. Yu. Approximate algorithms for scheduling parallelizable tasks. In *4th ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, 1992.

22 N. Vasić, M. Barisits, V. Salzgeber, and D. Kostić. Making cluster applications energy-aware. In *1st workshop on Automated Control for Datacenters and Clouds (ACDC)*, 2009.

23 P. Winkler and L. Zhang. Wavelength assignment and generalized interval graph coloring. In *14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 830–831, 2003.

24 A.J. Wood and B. Wollenberg. *Power Generation Operation and Control*. Wiley, 2nd edition, 1996.

25 Y. Zhang, X. Hu, and D.Z. Chen. Task scheduling and voltage selection for energy minimization. In *Design Automation Conference (DAC)*, pages 183–188, 2002.

A Near-linear Time Constant Factor Algorithm for Unsplittable Flow Problem on Line with Bag Constraints

Venkatesan T. Chakaravarthy, Anamitra R. Choudhury, and Yogish Sabharwal

IBM Research - India, New Delhi
{vechakra, anamchou, ysabharwal}@in.ibm.com

Abstract

Consider a scenario where we need to schedule a set of jobs on a system offering some resource (such as electrical power or communication bandwidth), which we shall refer to as bandwidth. Each job consists of a set (or *bag*) of job instances. For each job instance, the input specifies the start time, finish time, bandwidth requirement and profit. The bandwidth offered by the system varies at different points of time and is specified as part of the input. A feasible solution is to choose a subset of instances such that at any point of time, the sum of bandwidth requirements of the chosen instances does not exceed the bandwidth available at that point of time, and furthermore, at most one instance is picked from each job. The goal is to find a maximum profit feasible solution. We study this problem under a natural assumption called the no-bottleneck assumption (NBA), wherein the bandwidth requirement of any job instance is at most the minimum bandwidth available. We present a simple, near-linear time constant factor approximation algorithm for this problem, under NBA.

When each job consists of only one job instance, the above problem is the same as the well-studied unsplittable flow problem (UFP) on lines. A constant factor approximation algorithm is known for the UFP on line, under NBA. Our result leads to an alternative constant factor approximation algorithm for this problem. Though the approximation ratio achieved by our algorithm is inferior, it is much simpler, deterministic and faster in comparison to the existing algorithms. Our algorithm runs in near-linear time ($O(n \log^2 n)$), whereas the running time of the known algorithms is a high order polynomial. The core idea behind our algorithm is a reduction from the varying bandwidth case to the easier uniform bandwidth case, using a technique that we call *slicing*.

1998 ACM Subject Classification F.2.2 [Analysis of Algorithms]

Keywords and phrases Approximation Algorithms; Scheduling; Resource Allocation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.181

1 Introduction

We consider a general resource allocation problem in which we need to schedule jobs on a system offering a certain amount of some resource (such as electrical power, processing nodes, communication bandwidth). Each job consists of a set (or *bag*) of job instances, out of which at most one can be chosen. Each job instance requires a particular amount of the resource for its execution. The total amount of the resource offered by the system is different at different points of time. Our goal is to choose a subset of job instances such that at any timeslot, the total amount of resource requirement does not exceed the total amount of the resource available at that timeslot. We wish to maximize the profit of the

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 181–191

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

chosen subset of jobs. The problem formulation is motivated by its applications in cloud computing, bandwidth allocation in networks, smart energy management and allocating processor nodes on a multi-processor system. We refer to [4, 11, 1, 12, 8] for a discussion on some of these applications. Motivated by such scheduling and bandwidth allocation scenarios, we study an abstract problem called the *Varying bandwidth resource allocation problem with bag constraints* (BagVBRap), introduced in [8]. We use *bandwidth* as a generic term to refer to the resource under contention. The BagVBRap problem generalizes several previously studied scheduling and resource allocation problems. We next define the problem formally.

1.1 BagVBRap Problem Definition

The input consists of a set of jobs $\mathcal{J}$. Each job $J \in \mathcal{J}$ consists of a set of *job instances* of which at most one can be selected for execution. A instance u of a job J is specified by an interval $I_u = [a, b]$, where a and b are the *start time* and the *finish time* of the instance u; we assume that a and b are integers. The job instance u is also associated with a *bandwidth requirement* ρ_u and a profit p_u. Let D be the maximum finish time over all instances so that the interval associated with every job instance is contained in the span $[1, D]$. We refer to each integer $1 \leq t \leq D$ as a *timeslot*. For each timeslot t, the input specifies a number B_t which is the *bandwidth available* at timeslot t.

We use the term *instance* as a shorthand for job instance. Let $\mathcal{U}$ denote the set of all n instances over all the jobs in $\mathcal{J}$. For each instance $u \in \mathcal{U}$, we view each interval $I_u = [a, b]$ as a set of timeslots in the range $[a, b]$. We view each job as a *bag* of its instances. We say that the instance u is *active* at a timeslot t, if $t \in I_u$. For a timeslot t, let $A(t)$ denote the set of all instances active at timeslot t.

A feasible solution is a subset of instances $S \subseteq \mathcal{U}$ such that at every timeslot t, the sum of the bandwidth requirements of the instances from S active at time t is at most B_t, i.e, for every timeslot $1 \leq t \leq D$,

$$\sum_{u \in S \cap A(t)} \rho_u \leq B_t.$$

We call this the *bandwidth constraint*. Furthermore, it is required that at most one instance is picked from each job; we call this the *bag constraint*; we view a job as a bag of instances and hence the terminology. The problem is to find a feasible solution S such that the sum of the profits of the jobs in S is maximized. This completes the problem description.

The concept of bag constraints is quite powerful. Apart from handling the notion of release time and deadline, it can also work in a more general setting where a job can specify a set of possible time intervals where it can be scheduled. Moreover, BagVBRap allows for different instances of the same job to have different bandwidth requirements, processing times and profits.

The maximum and minimum available bandwidths over all timeslots will be of use in our discussion. We denote these by $\mathrm{B}_{\max}$ and $\mathrm{B}_{\min}$, that is $\mathrm{B}_{\max} = \max_{t \in [1,D]} B_t$ and $\mathrm{B}_{\min} = \min_{t \in [1,D]} B_t$. Similarly, the maximum and minimum bandwidth requirement over all instances is also of interest. We denote these by $\rho_{\max}$ and $\rho_{\min}$. That is, $\rho_{\max} = \max_{u \in \mathcal{U}} \rho_u$ and $\rho_{\min} = \min_{u \in \mathcal{U}} \rho_u$.

Remark: We can assume that for each instance u, the start time and end time of u are in the range $[1, 2n]$, since this leaves the problem combinatorially unchanged. Thus, we can assume that $D \leq 2n$.

1.2 Prior Work

The BagVBRap problem is a generalized formulation that captures as special cases many well-studied scheduling and resource allocation problems. Here we shall describe some important special cases and then present a brief survey of some of the prior work dealing with these problems.

- *Uniform bandwidth resource allocation problem (*UBRap*):* This is the special case of the BagVBRap problem, where the bandwidth available is uniform across all timeslots and the bag constraints do not exist. Meaning, each job consists of only one instance and for all $1 \leq t \leq D$, $B_t = B$ for some fixed B given as part of the input.
- *Uniform bandwidth resource allocation problem with bag constraints (*BagUBRap*):* This is the special case of the BagVBRap problem, where the bandwidth available is uniform across all timeslots.
- *Varying bandwidth resource allocation problem (*VBRap*):* This is the special case of the BagVBRap problem, where each job has only one instance. The VBRap problem is the same as the unsplittable flow problem (UFP) on line graphs, a well-studied problem.

Calinescu et al. [7] presented a 3-approximation for the UBRap problem, based on LP rounding technique that they refer to as "listing algorithm". Independently, Bar-Noy et al. [4] also presented a local-ratio based 3-approximation algorithm for the same problem. They also show how to handle bag constraints and derive a 5-approximation algorithm for the BagUBRap problem. A further special case of the BagUBRap problem is obtained when the bandwidth requirements and the bandwidth available are all unit. This special case has been studied under the name *weighted job interval selection problem* (WJISP), for which Bar-Noy et al. [5] and, independently, Berman and Dasgupta [6] presented local-ratio based 2-approximation algorithms.

For the VBRap problem (i.e., the UFP problem on line) Chakrabarti et al. [9] presented an algorithm with an approximation ratio of $O(\log(\rho_{\max}/\rho_{\min}))$. Bansal et al. [3] presented an $O(\log n)$-approximation algorithm for the same problem. No polynomial time constant factor approximation algorithm is known for this problem. However, in a break-through result, Bansal et al. [2] obtained a quasi-PTAS for UFP on line. For the BagVBRap problem, an $O(\log(\mathrm{B}_{\max}/\mathrm{B}_{\min}))$-approximation algorithm was obtained in [8]; this was achieved by extending the LP based "listing" algorithm of Calinescu et al. [7],

Obtaining a polynomial time constant factor approximation algorithm for the VBRap problem has remained a challenging open problem. However, this has been achieved under a reasonable assumption known as the *no-bottleneck assumption* (NBA).

No Bottleneck Assumption (NBA): We say that an input to the BagVBRap problem satisfies the *no bottleneck assumption (NBA)*, if the maximum bandwidth requirement of every job instance is less than the minimum bandwidth available. That is, $\rho_{\max} \leq \mathrm{B}_{\min}$.

Chakrabarti et al. [9] obtained the first constant factor approximation algorithm for the VBRap problem, under NBA. For the same special case, Chekuri et al. [10] improved the constant factor to $(2 + \epsilon)$. Both these algorithms are based on randomized rounding of LP solutions.

1.3 Our Result and Discussion

Our main result is a constant factor approximation algorithm for the BagVBRap problem, under NBA. The running time of the algorithm is $O(n \log^2 n)$, where n is the number of job instances. The approximation ratio is 120.

An important feature of our approach is the simplicity of both the algorithm and analysis. We show how to handle the non-uniformity or the varying nature of the bandwidths available, via a reduction to the easier case of uniform bandwidths. Namely, we present a simple reduction from the BagVBRap problem with NBA to the BagUBRap problem. Given a BagVBRap input with n job instances, our algorithm produces a BagUBRap instance having at most $O(n \log n)$ job instances. We then invoke the known constant factor approximation algorithm for BagUBRap, due to Bar-Noy et al. [4], which is based on the local ratio technique, and runs in time $O(n \log n)$, where n is the number of input job instances. Thus, our algorithm for BagVBRap runs in time $O(n \log^2 n)$.

Our result yields a constant factor approximation algorithm for the UFP problem on line, with NBA. The approximation ratio is 120. As mentioned earlier, Chakrabarti et al. [9] and Chekuri et al. [10] have presented constant factor approximation algorithms for this problem. The algorithm of Chekuri et al. guarantees an approximation ratio of $(2 + \epsilon)$ (for any $\epsilon > 0$); this algorithm is based on randomized LP rounding. This algorithm offers a tradeoff between approximation ratio and running time. The best running time achievable within this framework is more than $O(n^{10})$. As the approximatio ratio approaches 2, the running time grows substantially. Though our algorithm is inferior in terms of approximation ratio, it is simpler, deterministic and runs in near-linear time ($O(n \log^2 n)$). We believe that our technique of reducing varying bandwidths to uniform bandwidths may find application in other scenarios as well.

Organization. The rest of the paper is devoted towards presenting our constant factor approximation algorithm for the BagVBRap problem with NBA. For a better exposition of our main ideas, in Section 3, we first consider a special case where the available bandwidths are all integral multiples of $\mathrm{B}_{\min}$. For this case, we present a constant factor approximation algorithm. In Section 4, we will take care of the technicalities in dealing with the general case and derive a constant factor approximation algorithm for the general case, which runs in time $O(n \log^2 n)$. Both the algorithms use reductions to BagUBRap.

2 Preliminaries

Here, we develop some notations used throughout the paper. For a set of instances $X \subseteq \mathcal{U}$ and a timeslot $t \in [1, D]$, let $\rho_X(t)$ denote the sum of bandwidth requirements of all instances in X that are active at timeslot t, i.e., $\rho_X(t) = \sum_{u \in X \cap A(t)} \rho_u$.

By a *bandwidth profile* P, we mean a function that specifies a bandwidth for each timeslot. If P and Q are two bandwidth profiles, we define $P - Q$ to be the profile R given by: $R(t) = \max\{0, P(t) - Q(t)\}$, for all timeslots t. For a bandwidth profile P and a constant c, we define $c \otimes P$ to be the profile P' given by: $P'(t) = c \cdot P(t)$, for all timeslots t. By a *uniform strip of bandwidth* b, we mean a profile which uniformly takes the bandwidth to be b, for all timeslots. We say that the profile P is an *integral profile*, if for all timeslots t, $P(t)$ is an integral multiple of $\mathrm{B}_{\min}$ (we allow $P(t) = 0$).

We say that a set of instances X *fits into a profile* P, if for all timeslots t, $\rho_X(t) \leq P(t)$. The input specifies a profile P_{in} given by: $P_{in}(t) = B_t$, for all timeslots t. We shall refer to P_{in} as the input profile. Note that a feasible solution is simply a set of instances S that fits into the input profile P_{in} and satisfies the bag constraints.

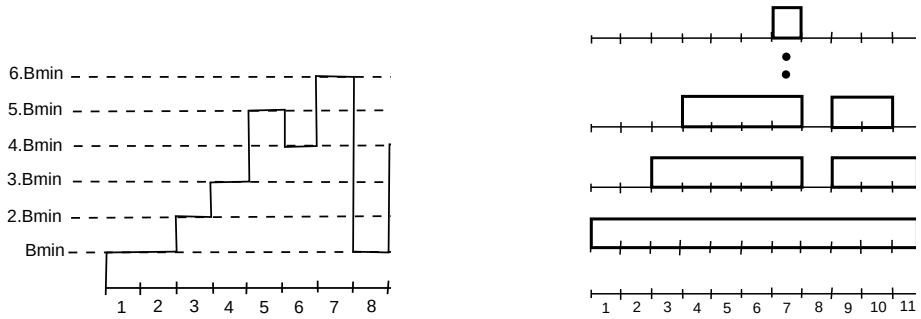


Figure 1 Illustration of slicing

3 BagVBRap with NBA: The Case of Integral Profiles

In this section, for a simple exposition of our main ideas, we focus on the special case where the input bandwidth profile P_{in} is integral. For this case, we present a constant factor approximation algorithm having running time $O(n^2 \log n)$. In the next section, we will take care of the technicalities in dealing with the non-integral input profiles and derive a constant factor approximation algorithm for the general case, which runs in time $O(n \log^2 n)$.

Let $K = \mathrm{B}_{\max}/\mathrm{B}_{\min}$; since the profile is assumed to be integral, K is an integer. We imagine the input bandwidth profile P_{in} to be a curve giving a value for each timeslot t. We slice the area under the curve horizontally into K *slices* each of height $\mathrm{B}_{\min}$, as illustrated in Figure 1. For $1 \leq j \leq K$, the bandwidth profile of the jth slice is defined as follows: for $1 \leq t \leq D$,

$$\mathrm{Slice_j(t)} \quad = \quad \begin{cases} \mathrm{B}_{\min} & \text{if } B_t \geq j \cdot \mathrm{B}_{\min} \\ 0 & \text{otherwise} \end{cases}$$

Consider a feasible solution $S \subseteq \mathcal{U}$ (which fits into the profile P_{in} and satisfies the bag constraints). The solution S is said to be *slice-respecting*, if it is possible to assign the instances of S to the K slices satisfying the following two properties: (i) each instance $u \in S$ is assigned to one of the K slices; (ii) for $1 \leq j \leq K$, the subset of instances assigned to the jth slice fits into the profile $\mathrm{Slice_j}$.

Let $S_{\mathrm{opt}} \subseteq \mathcal{U}$ be the optimum solution. Our algorithm and analysis have two main components:

- First, we will show that the optimum solution S_{opt} can be partitioned into 16 subsets such that each subset is a slice-respecting solution (see Section 3.1).
- Second, we will present an algorithm that will output a slice-respecting solution S such that the profit of S is a 5-approximation to the optimum slice-respecting solution (namely, the maximum profit solution among all slice respecting solutions). This algorithm is obtained via a reduction to the BagUBRap problem. (see Section 3.2).

It follows that S is a 80-approximation to S_{opt}. This yields the following theorem.

▶ **Theorem 1.** *There exists a* 80*-approximation algorithm for the* BagVBRap *problem with NBA when the input bandwidth profile P_{in} is integral. The running time of the algorithm is $O(n^2 \log n)$.*

3.1 Partitioning S_{opt} into Slice-respecting Solutions

In this section, we will prove the following lemma.

▶ **Lemma 2.** *Any feasible solution S can be partitioned into* 16 *subsets such that each subset is a slice-respecting solution.*

The intuitive idea behind the above claim is as follows. We will delete the first slice from the bottom of the profile and obtain a new profile. We shall identify a subset of instances Y and delete them such that the remaining instances fits into the new profile. We will show that the deleted subset of instances Y can be partitioned into a collection of 16 subsets such that each subset in the collection fits into the deleted (first) slice. We shall then apply the above procedure recursively K times obtaining K such collections. A slice-respecting solution can be formed by picking one subset from each collection. This way, we can form 16 slice respecting solutions.

The rest of the section is devoted to proving Lemma 2 formally. The following two lemmas are useful for this purpose.

▶ **Lemma 3.** *Let P be any integral profile. Let $X \subseteq \mathcal{U}$ be any subset of instances that fits into the profile P. Then, there exists a subset $Y \subseteq X$ such that: (i) for all timeslots t, $\rho_Y(t) \geq \min\{\rho_X(t), \mathrm{B}_{\min}\}$; (ii) Y fits into the uniform strip of bandwidth $4 \cdot \mathrm{B}_{\min}$.*

Proof. For a subset $T \subseteq \mathcal{U}$, a job instance is said to be *critical* for T if removal of the job instance from T causes the total bandwidth of the remaining jobs in T to fall below $\mathrm{B}_{\min}$ at some timeslot. More formally, a job instance u is said to be critical for T, if $\rho_{T\setminus\{u\}}(t) < \mathrm{B}_{\min}$ for some $t \in I_u$.

We start with $Y = X$ and then repeatedly remove jobs that are not critical for Y until no more such jobs exist. We argue that the remaining jobs in Y satisfy the required properties. The first property follows from the fact that we never remove a critical job. The second property is proved by contradiction. Suppose that for some timeslot $\tilde{t}$, $\rho_Y(\tilde{t}) > 4 \cdot \mathrm{B}_{\min}$. Let

$$t_l = \max_t\{t \leq \tilde{t} \text{ and } \rho_Y(t) < 2 \cdot \mathrm{B}_{\min}\} \quad \text{and} \quad t_r = \min_t\{t \geq \tilde{t} \text{ and } \rho_Y(t) < 2 \cdot \mathrm{B}_{\min}\}.$$

First, let us suppose that both t_l and t_r exist. Note that by definition of t_l and t_r, we have that $\rho_Y(t) \geq 2 \cdot \mathrm{B}_{\min}$, for all $t \in (t_l, t_r)$. We will now argue that there exists some job instance $u \in Y$, such that I_u is contained in (t_l, t_r). If this were not so, then for any timeslot $t' \in (t_l, t_r)$, any job instance u' active at timeslot t' would also be active at timeslot t_l or at timeslot t_r. This implies that for any $t' \in (t_l, t_r)$, $\rho_Y(t') \leq \rho_Y(t_l) + \rho_Y(t_r) < 4 \cdot \mathrm{B}_{\min}$. This would contradict our hypothesis that $\rho_Y(\tilde{t}) > 4 \cdot \mathrm{B}_{\min}$. Therefore, there exists a job instance u such that I_u is contained in (t_l, t_r). This combined with the fact that $\rho_Y(t) \geq 2 \cdot \mathrm{B}_{\min}$, for all $t \in (t_l, t_r)$ and the NBA implies that u is not a critical job. This contradicts our construction of Y.

Now, let us assume that t_l does not exist. Then $\rho_Y(t) \geq 2 \cdot \mathrm{B}_{\min}$, for all $t \in [1, \tilde{t}]$. Let $u' \in Y$ be the job with the earliest finish time and let its finish time be $t_{u'}$. Note that $\rho_Y(t) \geq 2 \cdot \mathrm{B}_{\min}$ for all $t \in [1, t_{u'}]$. This fact along with the NBA assumption implies that u' is not a critical job. This contradicts our construction of Y.

The case when t_r does not exist is handled similar to the case of t_l not existing as above. This completes the proof of the lemma. □

▶ **Lemma 4.** *Let $X \subseteq \mathcal{U}$ be any subset of instances that fits into the uniform strip of bandwidth $4 \cdot \mathrm{B}_{\min}$. Then, X can be partitioned into a collection of* 16 *subsets $\{X_1, X_2, \ldots, X_{16}\}$ such that each subset X_i fits into the uniform strip of bandwidth $\mathrm{B}_{\min}$.*

Proof. We say that an instance $u \in X$ is *large*, if $\rho_u > \mathrm{B}_{\min}/2$; otherwise, u is said to be small. Let $X_\ell \subseteq X$ be the set of large instances and let X_s be the set of small instances. Let

us first focus on the set X_ℓ. Create 8 uniform strips of bandwidth B_{min}, named $P_1, P_2, \ldots, P_8$, called *buckets*. Arrange the instances in X_ℓ in a list in increasing order of their start times. Scan this list and for each instance u, add u to a bucket where it can fit without violating the bandwidth constraint. The crucial claim is that each instance u will fit into some bucket. To see this claim, consider the first instance u that does not fit into any bucket. Let the starting timeslot of u be t_0. Since we are dealing with large instances, each bucket contains exactly one large instance active at the timeslot t_0 . Let Y be the set of these instances. Their combined bandwidth at t_0 is $\rho_Y(t_0) > 4 \cdot B_{min}$. This is a contradiction since X is assumed to fit into a uniform strip of bandwidth $4 \cdot B_{min}$.

The argument for small tasks is similar. Consider 8 uniform strips of bandwidth B_{min}, Q_1, Q_2, ..., Q_8, which are referred to as buckets. Scan the small instances in the increasing order of their start times. For each instance u, add u to a bucket where it would fit without violating the bandwidth constraint. As before, we claim that every instance will fit into at least one bucket. To see this, suppose u be the first instance that does not fit into any bucket. Let the starting timeslot of u be t_0. For each $1 \leq i \leq 8$, let Y_i be the set of instances in Q_i active at timeslot t_0. Since u is small, $\rho_{Y_i}(t_0) > B_{min}/2$. Let Y be the union of $Y_1, Y_2, \ldots, Y_8$. Their combined bandwidth at t_0 is $\rho_Y(t_0) > 4 \cdot B_{min}$. This is a contradiction since X is assumed to fit into a uniform strip of bandwidth $4 \cdot B_{min}$.

The set of instances added to the 16 buckets $P_1, P_2, \ldots, P_8$ and $Q_1, Q_2, \ldots, Q_8$ are taken to be the required subsets $X_1, X_2, \ldots, X_{16}$. □

We now prove Lemma 2. Let $P_0 = P_{in}$ be the input profile and let $X_0 = S$ be the given solution. We first invoke Lemma 3 with P_0 and X_0 as inputs and obtain a set of instances Y_1. We then apply Lemma 4 to partition Y_1 into a collection of 16 subsets $\{Y_1^1, Y_1^2, \ldots, Y_1^{16}\}$. Note that each of these subsets Y_1^i fits into the profile of the first slice. We delete the first slice (bottom-most slice) from the profile P_0 and obtain a profile P_1 (formally, we set $P_1 = P_0 - \text{Slice}_1$). We also delete the set of instances Y_1 from X_0 and get a new set of instances $X_1 = X_0 - Y_1$. Observe that the set of instances X_1 fits into the profile P_1. This allows us to apply the above process starting with P_1 and X_1 as inputs. Overall, we will apply the above process iteratively K times.

Formally, for $j = 1$ to K, we do as follows:

- Invoke Lemma 3 with P_{j-1} and X_{j-1} as inputs and obtain a set of instances Y_j.
- Invoke Lemma 4 to partition Y_j into a collection of 16 subsets $\{Y_j^1, Y_j^2, \ldots, Y_j^{16}\}$. Note that each set Y_j^i fits into the profile of the jth slice.
- Define a new integral profile $P_j = P_{j-1} - \text{Slice}_j$.
- Define $X_j = X_{j-1} - Y_j$. The set of instances X_j fits into the profile P_j.

We now form 16 solutions: for $1 \leq i \leq 16$, let $Z_i = \cup_{j=1}^{K} Y_j^i$. Each set Z_i is a slice-respecting solution. This completes the proof of Lemma 2.

3.2 Approximating the Optimum Slice-respecting Solution

In this section, we shall present an algorithm for finding a 5-approximation to the optimum slice-respecting solution. We achieve this goal via a reduction to the BagUBRap problem, for which Bar-Noy et al. [4] designed a local-ratio based 5-approximation algorithm.

Let S^* be the optimum slice-respecting solution. Our goal is to find a 5-approximation to S^*, via a reduction to the BagUBRap problem. Let $\mathcal{I}$ be the input instance, which we will transform into an input instance $\widetilde{\mathcal{I}}$ of BagUBRap. Let the timespan of $\mathcal{I}$ be $[1, D]$ so that all the job instances finish before D. In the transformed instance $\widetilde{\mathcal{I}}$, the timespan will be $[1, KD]$, where K is the number of slices. The bandwidth profile of $\widetilde{\mathcal{I}}$ is obtained by concatenating

the bandwidth profiles of the K slices; namely, for $1 \leq j \leq K$, the range $[1+(j-1)D, jD]$ corresponds to the jth slice. Formally, in the input instance $\widetilde{I}$, the bandwidth available $\widetilde{B}_t$ is declared as follows: for $1 \leq j \leq K$ and for each timeslot $t \in [1+(j-1)D, jD]$, we set $\widetilde{B}_t = \mathrm{Slice_j(t-(j-1)D)}$. Note that in $\widetilde{\mathcal{I}}$, the bandwidths available at all timeslots are either $\mathrm{B_{min}}$ or 0.

For each job $J \in \mathcal{J}$ of the input $\mathcal{I}$, we create a corresponding job $\widetilde{J}$ in $\widetilde{\mathcal{I}}$. For each job instance $u \in \mathcal{U}$ in the input instance $\mathcal{I}$ with associated interval $I_u = [a, b]$, we create at most K copies of u as follows. For each slice $1 \leq j \leq K$, we create a copy of u, if the instance u can be scheduled in the jth slice (meaning, for all $t \in [a, b]$, $\mathrm{Slice_j(t)} \neq 0$); this copy of u is declared to have the interval $I_u^j = [(j-1)D + a, (j-1)D + b]$. This way, at most K copies are created for the instance u. Each of these copies will have the same bandwidth requirement and profit as the instance u. If $J \in \mathcal{J}$ is the job to which the instance u belongs, then all these copies of u are made instances of the corresponding job $\widetilde{J}$ in the transformed input $\widetilde{\mathcal{I}}$. For instance, if a job $J \in \mathcal{J}$ had d instances in $\mathcal{I}$, its corresponding job in $\widetilde{\mathcal{I}}$ would have at most dK instances.

A minor issue in the above construction is that in $\widetilde{\mathcal{I}}$, the bandwidths available are not uniform across all the timeslots. However, these are all either $\mathrm{B_{min}}$ or 0. This is easily addressed via compressing the profile by deleting all timeslots where the bandwidth availability is 0. This way, we get a BagUBRap instance. This completes the reduction.

It is easy to see that a slice-respecting feasible solution S to $\mathcal{I}$ can be converted into a feasible solution $\widetilde{S}$ for $\widetilde{\mathcal{I}}$ and vice versa. To see the forward direction, if u is an instance picked in the solution S and assigned to slice j, we pick the instance I_u^j in $\widetilde{S}$. For the reverse direction, if the instance I_u^j is picked in the solution $\widetilde{S}$, we include the instance I_u in S and assign it to the slice j. Now invoking the 5-approximation algorithm for the BagUBRap problem [4], we get the desired 5-approximation to the optimum slice-respecting solution.

The 5-approximation algorithm for the BagUBRap problem runs in time $O(\widetilde{n}\log(\widetilde{n}))$, where $\widetilde{n}$ is the number of job instances in $\widetilde{\mathcal{I}}$. In our reduction, for each job instance $u \in \mathcal{I}$, at most K instances are created in $\widetilde{\mathcal{I}}$ and hence, $\widetilde{n} \leq Kn$. Recall that $K = \mathrm{B_{max}}/\mathrm{B_{min}}$. We can assume that $\mathrm{B_{max}} \leq n\rho_{\max}$. Under NBA, we also have that $\mathrm{B_{min}} \geq \rho_{\max}$. Hence, $K = O(n)$. Thus, our algorithm runs in time $O(n^2 \log n)$.

4 A Constant Factor Approximation for the General Case

We now consider the general case involving non-integral input profiles and derive a constant factor approximation algorithm having running time $O(n \log^2 n)$. The core idea of slicing is the same as that of the earlier section. The new result is obtained by dealing with some technicalities. First, instead of slicing the input profile into slices of equal height, here we shall slice the profile into slices of geometrically increasing heights. This enables us to bring down the running time. Secondly, we shall "floor" the input profile by carefully decreasing the bandwidth available at every timeslot so that it becomes "geometric" and it is therefore, suitable for "geometric slicing". However, the "flooring" increases the approximation ratio by a constant factor.

We say that a profile P is a *geometric integral profile* (GIP), if one of the following two conditions is true:

- For all timeslots t, either $P(t) = 0$ or $P(t) = 2^i \cdot \mathrm{B_{min}}$, for some integer $i \geq 0$.
- There exists an integer constant $c \geq 0$ such that for all timeslots t, either $P(t) = 0$ or $P(t) = (2^i - 2^c)\mathrm{B_{min}}$, for some integer $i \geq 0$.

In the former case the profile is said to be a *type-1* GIP and in the latter case, the profile

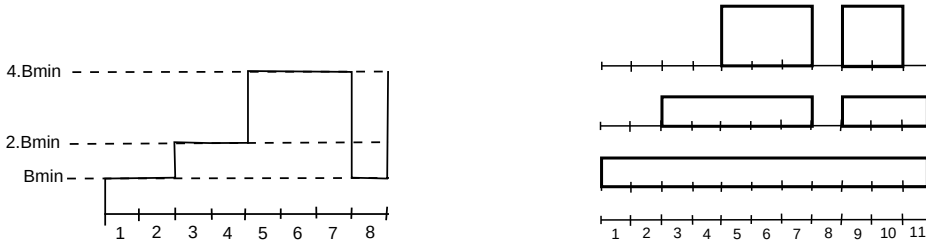


Figure 2 Illustration of slicing

said to be a *type-2* GIP. The idea of having two different types of GIPs is that we shall start with a type-1 GIP and as we delete slices iteratively, we obtain type-2 GIPs.

We first convert the input profile P_{in} into a type-1 GIP by "flooring" the bandwidths, as follows. Define a new profile $\widetilde{P}_{in}$ given by $\widetilde{P}_{in}(t) = 2^i \cdot \mathrm{B}_{\min}$, where $i = \lfloor \log(P_{in}(t)/\mathrm{B}_{\min}) \rfloor$. Figure 2 presents the flooring of the profile shown in Figure 1. A feasible solution $S \subseteq \mathcal{U}$ (which fits into the profile P_{in}) may not fit into the profile $\widetilde{P}_{in}$. Our algorithm will actually output a solution that fits into the profile $\widetilde{P}_{in}$. In this process, we lose a constant factor in the approximation.

Similar to the previous section, we now define the notion of slices and slice-respecting solutions. Let $K = 1 + \lfloor \log(\mathrm{B}_{\max}/\mathrm{B}_{\min}) \rfloor$. We slice the profile $\widetilde{P}_{in}$ horizontally into K *slices* in a geometric manner. The profile of the first slice Slice_1 is the uniform strip of bandwidth $\mathrm{B}_{\min}$. For $2 \leq j \leq K$, the bandwidth profile of the jth slice is given by the following profile: for $1 \leq t \leq D$,

$$\mathrm{Slice_j(t)} \quad = \quad \begin{cases} 2^{j-2} \cdot \mathrm{B}_{\min} & \text{if } \widetilde{P}_{in}(t) \geq 2^{j-1} \cdot \mathrm{B}_{\min} \\ 0 & \text{otherwise} \end{cases}$$

See Figure 2 for an illustration of the slices.

Consider a feasible solution $S \subseteq \mathcal{U}$ (which fits into the profile P_{in} and satisfies the bag constraints). The solution S is said to be *slice-respecting*, if it is possible to assign the instances of S to the K slices satisfying the following two properties: (i) each instance $u \in S$ is assigned to one of the K slices; (ii) for $1 \leq j \leq K$, the subset of instances assigned to the jth slice fits into the profile $\mathrm{Slice_j}$.

Let $S_{\mathrm{opt}} \subseteq \mathcal{U}$ be the optimum solution. Our algorithm and analysis have two main components:

- First, we will show that the solution S_{opt} can be partitioned into 24 subsets such that each subset is a slice-respecting solution.
- Second, we will present an algorithm that will output a slice-respecting solution S such that the profit of S is a 5-approximation to the optimum slice-respecting solution.

It follows that S is a 120-approximation to S_{opt}. This yields the following theorem.

▶ **Theorem 5.** *There exists a* 120*-approximation algorithm for the* BagVBRap *problem with NBA. The running time of the algorithm is* $O(n \log^2 n)$.

We now focus on the first component and prove a lemma similar in spirit to Lemma 2.

▶ **Lemma 6.** *Any feasible solution* $S \subseteq \mathcal{U}$ *can be partitioned into* 24 *subsets such that each subset is a slice-respecting solution.*

We first state two variants of Lemma 3 that deal with the two types of GIPs. The proofs are similar to that of Lemma 3.

▶ **Lemma 7.** *Let P be any type-1* GIP*. Let $X \subseteq \mathcal{U}$ be any subset of instances that fits into the profile $2 \otimes P$. Then, there exists $Y \subseteq X$ such that: (i) for all timeslots t, $\rho_Y(t) \geq \min\{\rho_X(t), 2 \cdot \mathrm{B_{min}}\}$; (ii) Y fits the uniform strip of bandwidth $6 \cdot \mathrm{B_{min}}$.*

▶ **Lemma 8.** *Let P be any type-2* GIP *with parameter $c \geq 0$. Let $X \subseteq \mathcal{U}$ be any subset of instances that fits into the profile $2 \otimes P$. Then, there exists $Y \subseteq X$ such that: (i) for all timeslots t, $\rho_Y(t) \geq \min\{\rho_X(t), 2 \cdot 2^c \cdot \mathrm{B_{min}}\}$; (ii) Y fits into the uniform strip of bandwidth $6 \cdot 2^c \cdot \mathrm{B_{min}}$.*

We next state a variant of Lemma 4. The proof is similar to that of Lemma 4.

▶ **Lemma 9.** *Let $X \subseteq \mathcal{U}$ be any subset of instances that fits into the uniform strip of bandwidth $6 \cdot \alpha \cdot \mathrm{B_{min}}$, for some integer $\alpha \geq 1$. Then, X can be partitioned into 24 subsets $X_1, X_2, \ldots, X_{24}$ such that each subset X_i fits into the uniform strip of bandwidth $\alpha \cdot \mathrm{B_{min}}$.*

We now prove Lemma 6. The proof is similar to that of Lemma 2 and is proved in an iterative manner. In the first iteration, we apply Lemma 7 and Lemma 9 on profile $\widetilde{P}_{in}$, which is a type-1 GIP. In the subsequent iterations, we will apply Lemma 8 and 9, as the profiles considered in these iterations are type-2 GIPs.

Let $P_0 = \widetilde{P}_{in}$ be the input profile, which is a type-1 GIP. Let $X_0 = S$ be the given solution that fits into the profile P_{in}. Observe that X_0 fits into the profile $2 \otimes P_0$. We first invoke Lemma 7 with P_0 and X_0 as inputs and obtain a set of instances Y_1. We then apply Lemma 9 to partition Y_1 into 24 subsets $Y_1^1, Y_1^2, \ldots, Y_1^{24}$ (here $\alpha = 1$). Note that each of these subsets Y_1^i fits into the profile Slice_1. We delete the first slice (bottom-most slice) from the profile P_0 and obtain a profile P_1 (formally, we set $P_1 = P_0 - \mathrm{Slice}_1$). We also delete the set of instances Y_1 from X_0 and get a new set of instances $X_1 = X_0 - Y_1$.

The profile P_1 is a type-2 GIP with parameter $c = 0$ and X_1 fits into $2 \otimes P_1$. We invoke Lemma 8 with P_1 and X_1 as inputs and obtain a set of instances Y_2. We then apply Lemma 9 to partition Y_2 into 24 subsets $Y_2^1, Y_2^2, \ldots, Y_2^{24}$ (here $\alpha = 2^0 = 1$). Note that each of these subsets Y_2^i fits into the profile Slice_2. We now delete the second slice from the profile P_1 and obtain a profile P_2 (formally, we set $P_2 = P_1 - \mathrm{Slice}_2$). We also delete the set of instances Y_2 from X_1 and get a new set of instances $X_2 = X_1 - Y_2$. The profile P_2 is a type-2 GIP with parameter $c = 1$ and X_2 fits into $2 \otimes P_2$.

We can now iteratively apply the above process starting with P_2 and X_2 as inputs. We continue like this for K iterations. Formally, for $j = 2$ to K, do as follows:

- Invoke Lemma 8 with P_{j-1} and X_{j-1} as inputs and obtain a set of instances Y_j.
- Invoke Lemma 9 to partition Y_j into 24 subsets $Y_j^1, Y_j^2, \ldots, Y_j^{24}$. Each set Y_j^i fits into the profile $\mathrm{Slice_j}$.
- Define a profile P_j given by $P_j = P_{j-1} - \mathrm{Slice_j}$. The profile P_j is a type-2 GIP with parameter $c = j - 2$.
- Define $X_j = X_{j-1} - Y_j$. The set of instances X_j fits into the profile $2 \otimes P_j$.

We now form 24 solutions: for $1 \leq i \leq 24$, let $Z_i = \cup_{j=1}^{K} Y_j^i$. Each set Z_i is a slice-respecting solution. This completes the proof of Lemma 6.

We now outline the second component required to prove Theorem 5. What we need is an algorithm that approximates the optimum slice respecting solution within a factor of 5. The construction is the same as that of Section 3.2. However, we do not get a BagUBRap instance, since the bandwidths available are not uniform. Nevertheless, we can scale the

slices (and the associated job instances), so as to make all the slices of uniform bandwidth. This way we can get a BagUBRap instance.

In the current scenario $K = O(\log(B_{max}/B_{min}))$. Now an analysis similar to the one performed in the previous section shows that the running time of our algorithm is $O(n \log^2 n)$.

References

1. S. Albers. Resource Management in Large Networks. In J. Lerner, D. Wagner, and K. Zweig, editors, *Algorithmics for Large and Complex Networks: Design, Analysis, and Simulation*, pages 227–246. Springer Berlin/Heidelberg, 2009.
2. N. Bansal, A. Chakrabarti, A. Epstein, and B. Schieber. A quasi-PTAS for unsplittable flow on line graphs. In *ACM Symposium on Theory of Computing (STOC)*, pages 721–729, 2006.
3. N. Bansal, Z. Friggstad, R. Khandekar, and M. Salavatipour. A logarithmic approximation for unsplittable flow on line graphs. In *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 702–709, 2009.
4. A. Bar-Noy, R. Bar-Yehuda, A. Freund, J. Noar, and B. Schieber. A unified approach to approximating resource allocation and scheduling. *Journal of the ACM*, 48(5):1069–1090, 2001.
5. A. Bar-Noy, S. Guha, , J. Noar, and B. Schieber. Approximating the throughput of multiple machines in real-time scheduling. *Siam Journal of Computing*, 31(2):331–352, 2001.
6. P. Berman and B. Dasgupta. Multi-phase algorithms for throughput maximization for real-time scheduling. *Journal of Combinatorial Optimization*, 4:307–323, 2000.
7. G. Calinescu, A. Chakrabarti, H. Karloff, and Y. Rabani. Improved approximation algorithms for resource allocation. In *Prooceedings of the 9th International Conference on Interger Programming and Combinatorial Optimization*, 2002.
8. V. Chakaravarthy, V. Pandit, Y. Sabharwal, and D. Seetharam. Varying bandwidth resource allocation problem with bag constraints. In *24th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2010.
9. A. Chakrabarti, C. Chekuri, A. Gupta, and A. Kumar. Approximation algorithms for the unsplittable flow problem. *Algorithmica*, 47(1):53–78, 2007.
10. C. Chekuri, M. Mydlarz, and F. Shepherd. Multicommodity demand flow in a tree and packing integer programs. *ACM Transactions on Algorithms*, 3(3), 2007.
11. T. Erlebach and F. Spieksma. Interval selection: Applications, algorithms, and lower bounds. *Journal of Algorithms*, 46(1):27–53, 2003.
12. M. Flammini, G. Monaco, G. Moscardelli, H. Shachnai, M. Shalom, T. Tamir, and S. Zaks. Minimizing total busy time in parallel scheduling with application to optical networks. In *23rd IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2009.

Place-Boundedness for Vector Addition Systems with one zero-test*

Rémi Bonnet[1], Alain Finkel[1], Jérôme Leroux[2], and Marc Zeitoun[1,2]

1 **LSV, ENS Cachan, CNRS & INRIA, France.**
`firstname.lastname@lsv.ens-cachan.fr`
2 **LaBRI, Univ. Bordeaux & CNRS, France.**
`firstname.lastname@labri.fr`

Abstract

Reachability and boundedness problems have been shown decidable for Vector Addition Systems with one zero-test. Surprisingly, place-boundedness remained open. We provide here a variation of the Karp-Miller algorithm to compute a basis of the downward closure of the reachability set which allows to decide place-boundedness. This forward algorithm is able to pass the zero-tests thanks to a finer cover, hybrid between the reachability and cover sets, reclaiming accuracy on one component. We show that this filtered cover is still recursive, but that equality of two such filtered covers, even for usual Vector Addition Systems (with no zero-test), is undecidable.

Keywords and phrases Vector addition systems; Inhibitor arcs; Karp-Miller algorithms; Reachability sets; Cover sets; Well-quasi orders.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.192

1 Introduction

Context. Petri Nets, Vector Addition Systems (VAS), and Vector Addition Systems with control states (VASS) are equivalent well-known classes of counter systems for which the reachability problem is decidable [19, 17, 18], even if its complexity is still an open problem. On the other hand, testing equality of the reachability sets of two such systems is undecidable [12]. For that reason, one cannot compute a canonical finite representation of the reachability set that would make it possible to test for equality. However, there is such an effective finite representation for the *cover*, a useful over-approximation of the reachability set which is connected to various verification problems.

If we add to VAS the ability to test at least two counters to zero, one obtains a model equivalent to Minsky machines, for which all nontrivial properties are undecidable. The study of VAS with *a single* zero-test transition began recently, and very few results are known for this model. Reinhardt [21] has shown that the reachability problem is decidable for VASS with one zero-test transition (as well as for hierarchical zero-tests). Abdulla and Mayr have shown that the coverability problem is decidable in [2], by using the backward procedure of Well Structured Transition Systems [1]. See [10] for a survey. The boundedness problem (whether the reachability set is finite), the termination and the reversal-boundedness problem (whether the counters can alternate infinitely often between the increasing and the decreasing modes) are all decidable by using a forward procedure, a finite but *non-complete* Karp and Miller tree [9]. The place-boundedness problem, and more generally the possibility to

* Supported by the Agence Nationale de la Recherche, AVERISS (grant ANR-06-SETIN-001) and AVERILES (grant ANR-05-RNTL-002).

compute a finite representation of the cover were still open problems. Only in the particular case of dimension 2 with control states, the reachability set is semilinear and its basis and periods are computable [11] and then the place-boundedness is decidable; but this result cannot be extended in dimension 3, even without zero-test [14].

Our contribution. We give an algorithm for computing a finite representation of the cover for a VAS with one zero-test. This result makes it possible to decide the place-boundedness, which is in general undecidable for VAS extensions (such as VAS with resets [5] or Lossy Minsky machines, *i.e.* Lossy VAS with zero-test transitions [3, 20]). Our proof techniques introduce a filtered cover, an hybrid between the reachability and cover sets, which unlike the cover reclaims *accuracy* on one component. We show that this set is recursive, but that one cannot decide the equality of such filtered covers of two VAS (even without zero-test). Thus, our work is a contribution to understanding the limits of decidability, taking into account two parameters: the models (VAS and VAS with zero-test) and the problems (reachability, cover and filtered cover).

The difficulty. The central problem is to compute the cover of a VAS *with one zero-test*. Let us explain the reasons why the usual Karp and Miller is not sufficient for that purpose. A natural idea appearing in [9] is to adapt the classical Karp-Miller construction [15], first building the Karp-Miller tree, but *without* firing the zero test. To continue the construction after this first stage, we need to fire the zero test from the leaves of the Karp-Miller tree carrying a 0 value on the component tested to 0. The problem is that accelerations performed while building the Karp-Miller tree may have produced, on this component in the label of such a leaf, an ω value which represents infinity, and abstracts actual values. For that reason, one may not be able to determine if the zero test succeeds or not. We therefore want a more accurate information for the labeling of the leaves, for the component tested to 0. This is what the filtered cover actually captures.

The schema of our proof.

1. We start in Section 3 with usual VAS: we extend the decidability of the reachability problem for VAS, in proving that the set $\mathsf{Lim\,Reach}$ of *limits* of increasing sequences of reachable states is also recursive ($\mathsf{Lim\,Reach}$ contains the reachability set). The set $\mathsf{Lim\,Reach}$ is a more sophisticated set than both the cover and the reachability set. It allows us to know whether an element in $(\mathbb{N}\cup\{\omega\})^d$ is a reachable state or is the limit of a sequence of reachable states. This information is not given by the reachability set neither by the cover. The proof carries on by using Higman's Lemma, using a nontrivial ordering.
2. In Section 4, we refine the definition of the cover in which the first component has now to be exactly known (and not only bounded by a maximum). We prove that, for VAS, a finite basis of this filtered cover is *still computable* by using the recursivity of $\mathsf{Lim\,Reach}$.
3. We finally compute in Section 5 the finite basis of the cover of a VAS with one zero-test by using a variation of the Karp and Miller algorithm that uses the previously defined filtered covers in order to convey enough information to go through the zero-test.

Due to lack of space, some proofs are omitted.

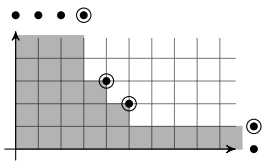
2 Vector Addition Systems

Orderings and vectors. An *ordering* $\preccurlyeq$ on a set X is a reflexive, transitive and antisymmetric binary relation on X. Given $x, y \in X$, we write $x \prec y$ for $x \preccurlyeq y$ and $x \neq y$. For $d \geqslant 1$,

we write any $\boldsymbol{x} \in X^d$ as $\boldsymbol{x} = (\boldsymbol{x}(1), \ldots, \boldsymbol{x}(d))$, with $\boldsymbol{x}(i) \in X$. The *pointwise ordering* on X^d, still denoted $\preccurlyeq$, is defined by $\boldsymbol{x} \preccurlyeq \boldsymbol{y}$ if $\boldsymbol{x}(i) \preccurlyeq \boldsymbol{y}(i)$ for all i. For $\boldsymbol{x_1} \in X^{d_1}$ and $\boldsymbol{x_2} \in X^{d_2}$, we let $(\boldsymbol{x_1}, \boldsymbol{x_2}) \in X^{d_1+d_2}$ be the vector obtained by gluing $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$. For $X = \mathbb{N}$, let $\mathbf{0}$ be the vector whose components are all 0, and for $i \in \{1, \ldots, d\}$, let $\boldsymbol{e_i}$ be the vector such that $\boldsymbol{e_i}(i) = 1$ and $\boldsymbol{e_i}(k) = 0$ if $k \neq i$. Finally, given $Y \subseteq X$, let $\downarrow_{\preccurlyeq} Y = \{x \in X \mid \exists y \in Y,\ x \preccurlyeq y\}$ denote the downward closure of Y with respect to $\preccurlyeq$. The set Y is said *downward closed* if $Y = \downarrow_{\preccurlyeq} Y$. When working in $\mathbb{N}^d$ or $\mathbb{N}_\omega^d$ (see below) we shorten the downward closure operator $\downarrow_{\leqslant}$ as $\downarrow$.

Downward closed sets of $\mathbb{N}^d$. Given an ordered set, one may under suitable hypotheses construct a topological completion of this set to recover a *finite description* of downward closed sets [7, 8]. The completion of $\mathbb{N}^d$ is $\mathbb{N}_\omega^d$, with $\mathbb{N}_\omega = \mathbb{N} \cup \{\omega\}$, where we extend $\leqslant$ by $n \leqslant \omega$ for all $n \in \mathbb{N}_\omega$. The results of [7, 8] in this case yield that, if $D \subseteq \mathbb{N}^d$ is downward closed, then $D = \mathbb{N}^d \cap \downarrow B$ for some finite set $B \subseteq \mathbb{N}_\omega^d$, which we call a (finite) *basis* of D. One can show that the maximal elements of any basis B of D still form a basis which does not depend of B. It is minimal for inclusion among all basis, and is called the *minimal basis*.

An example. Let us consider in $\mathbb{N}^2$ the downward closed set $\big\{(x,y) \in \mathbb{N}^2 \mid x \leqslant 3 \vee y \leqslant 1\big\} \cup \big\{(4,2),(4,3),(5,2)\big\}$. A (non-minimal) basis is $(\{0,1,2,3\} \times \{\omega\}) \cup \{(4,3),(5,2)\} \cup \{\omega\} \times \{0,1\}$. It is shown with dots $\bullet$ in the figure, where elements involving ω fall beyond the grid. The elements of the minimal basis are circled.



▶ **Definition 1. ($\mathbf{VAS_0}$).** A *Vector Addition System of dimension d with one zero-test (VAS_0)* is a tuple $\langle A, a_Z, \delta, \boldsymbol{x}_{\text{in}} \rangle$, where A is a finite alphabet of *actions*, $a_Z \notin A$ is called the *zero-test*, $\delta : A \cup \{a_Z\} \to \mathbb{Z}^d$ is a mapping, and $\boldsymbol{x}_{\text{in}} \in \mathbb{N}^d$ is the *initial state*.

Intuitively, a VAS_0 works on d counters, one for each component, whose initial values are given by $\boldsymbol{x}_{\text{in}}$. Executing $a \in A \cup \{a_Z\}$ translates the counters according to $\delta(a) \in \mathbb{Z}^d$. The mapping δ extends to a monoid morphism $\delta : (A \cup \{a_Z\})^* \to \mathbb{Z}^d$, so that $\delta(\varepsilon) = \mathbf{0}$ and $\delta(uv) = \delta(u) + \delta(v)$ for $u, v \in (A \cup \{a_Z\})^*$. A word $u \in (A \cup \{a_Z\})^*$ is *fireable from* $\boldsymbol{x} \in \mathbb{N}^d$ if

(*a*) for every prefix v of u, we have $\boldsymbol{x} + \delta(v) \geqslant \mathbf{0}$, and
(*b*) for every prefix wa_Z of u, we have $[\boldsymbol{x} + \delta(w)](1) = 0$.

The first condition means that all counters must remain nonnegative while firing actions. The second one says that the zero-test a_Z is possible only when the first counter is zero. We write $\boldsymbol{x} \xrightarrow{u} \boldsymbol{y}$ if u is fireable from $\boldsymbol{x}$ and $\boldsymbol{y} = \boldsymbol{x} + \delta(u)$. This implies in particular that $\boldsymbol{x}, \boldsymbol{y} \geqslant \mathbf{0}$.

▶ **Definition 2. (VAS).** A *Vector Addition System (VAS) of dimension d* is a tuple $\langle A, \delta, \boldsymbol{x}_{\text{in}} \rangle$ where A is a finite alphabet, $\delta : A \to \mathbb{Z}^d$ is a mapping, and $\boldsymbol{x}_{\text{in}} \in \mathbb{N}^d$ is the *initial state*.

A VAS is a particular VAS_0: choosing $\delta(a_Z) = -\boldsymbol{e}_1$ makes the zero-test a_Z never fireable. Given the VAS $\mathcal{S} = \langle A, \delta, \boldsymbol{x}_{\text{in}} \rangle$, we say that $u \in A^*$ is *fireable* if condition (*a*) above is satisfied.

For a VAS_0 or a VAS $\mathcal{S}$, the *reachability set* $\mathsf{Reach}(\mathcal{S})$ and the *cover* $\mathsf{Cover}(\mathcal{S})$ of $\mathcal{S}$ are:

$$\mathsf{Reach}(\mathcal{S}) = \{\boldsymbol{x}_{\text{in}} + \delta(u) \mid u \text{ is fireable in } \mathcal{S}\},$$
$$\mathsf{Cover}(\mathcal{S}) = \downarrow \mathsf{Reach}(\mathcal{S}).$$

We call elements of $\mathsf{Reach}(\mathcal{S})$ reachable *states* (also called reachable markings in related work). The reachability (resp. coverability) problem consists in deciding membership in

the set $\mathsf{Reach}(\mathcal{S})$ (resp. in $\mathsf{Cover}(\mathcal{S})$). Reachability is decidable for VAS [19, 17, 18] and VAS_0 [21].

▶ **Theorem 3.** *Given a VAS $\mathcal{S}$, the reachability problem is decidable.*

Testing membership in the cover set is easier. One even gets a more precise result [15, 10, 8].

▶ **Theorem 4.** *Given a VAS $\mathcal{S}$, one can effectively compute a finite basis of* $\mathsf{Cover}(\mathcal{S})$.

Observe that from a finite basis B of a downward closed set D, one can effectively test membership in D. Therefore, one can effectively test membership in $\mathsf{Cover}(\mathcal{S})$. Computing a finite basis of the cover makes it possible to decide place-boundedness, that is, whether the projection of $\mathsf{Reach}(\mathcal{S})$ on some given component is bounded. In the next sections, we will show that one can also effectively compute a finite basis for the cover of a VAS_0.

3 Limits of reachable states of a VAS

Limits in $\mathbb{N}_\omega^d$. A sequence $(\ell_n)_{n\geqslant 0}$ (also written $(\ell_n)_n$) of elements of $\mathbb{N}_\omega$ *has limit* $\ell \in \mathbb{N}_\omega$, noted $\lim_n \ell_n = \ell$, if either it is ultimately constant with value ℓ, or its subsequence of integer values is infinite, it tends to infinity, and $\ell = \omega$. A sequence $(\boldsymbol{x_n})_n$ of vectors of $\mathbb{N}_\omega^d$ has limit $\boldsymbol{x} \in \mathbb{N}_\omega^d$, noted $\lim_n \boldsymbol{x_n} = \boldsymbol{x}$, if $\lim_n \boldsymbol{x_n}(i) = \boldsymbol{x}(i)$ for all $i \in \{1, \ldots, d\}$.

For $M \subseteq \mathbb{N}_\omega^d$, we denote by $\mathsf{Lim}\, M$ the set of limits of sequences of elements of M. Note that $M \subseteq \mathsf{Lim}\, M$. Topologically speaking, $\mathsf{Lim}\, M$ is the least closed set (for the topology associated with the ordering) containing M and is usually called the (topological) closure of M. Also note that for $M \subseteq \mathbb{N}^d$, if $\mathsf{Lim}\, M$ is recursive, then so is $M = \mathbb{N}^d \cap \mathsf{Lim}\, M$. However, in general, M may be recursive while $\mathsf{Lim}\, M$ is not.

We prove in this section the following statement.

▶ **Theorem 5.** $\mathsf{Lim}\,\mathsf{Reach}(\mathcal{S})$ *is recursive.*

We do so by proving that $\mathsf{Lim}\,\mathsf{Reach}(\mathcal{S})$ and its complement in $\mathbb{N}_\omega^d$ are both recursively enumerable. We start by proving that $\mathsf{Lim}\,\mathsf{Reach}(\mathcal{S})$ is recursively enumerable, by introducing *productive sequences*, a notion inspired by Hauschildt [13].

▶ **Definition 6.** Let $\mathcal{S} = \langle A, \delta, \boldsymbol{x}_{\mathrm{in}}\rangle$ be a VAS. A sequence $\pi = (u_i)_{0\leqslant i\leqslant k}$ of words $u_i \in A^*$ is *productive* in $\mathcal{S}$ for a word $v = a_1 \cdots a_k$ $(a_i \in A)$ if

(1) the partial sums $\delta(u_0) + \cdots + \delta(u_i)$ are nonnegative for every $i \in \{0, \ldots, k\}$, and
(2) the word $u_0 a_1 u_1 \cdots a_k u_k$ is fireable from $\boldsymbol{x}_{\mathrm{in}}$.

The total sum $\sum_{i=0}^{k} \delta(u_i)$ is called the *production* of π and is simply denoted $\delta(\pi)$.

The following lemma provides a characterization of the productive sequences.

▶ **Lemma 7.** *A sequence $\pi = (u_i)_{0\leqslant i\leqslant k}$ is productive for $v = a_1 \cdots a_k$ if and only if the words $u_0^n a_1 u_1^n \cdots a_k u_k^n$ are fireable from $\boldsymbol{x}_{in}$ for all $n \geqslant 1$. In particular, every marking $\boldsymbol{x}_{in} + \delta(v) + n\delta(\pi)$ where $n \geqslant 1$ is reachable from $\boldsymbol{x}_{in}$.*

Proposition 9 below shows that limits of reachable states can be witnessed by productive sequences. Its essential argument is Higman's Lemma. Recall that an ordering $\preccurlyeq$ is *well* if every infinite sequence $(\ell_n)_{n\in\mathbb{N}}$ admits an infinite increasing subsequence $(\ell_{n_k})_{k\in\mathbb{N}}$, *i.e.*, such that $\ell_{n_0} \preccurlyeq \ell_{n_1} \preccurlyeq \ell_{n_2} \preccurlyeq \cdots$. The pointwise ordering on $\mathbb{N}^d$ or on $\mathbb{N}_\omega^d$ is well (Dickson's Lemma).

Higman's Lemma. For a (possibly infinite) set Σ, we denote by Σ^* the set of finite words over Σ. Given an ordering $\preccurlyeq$ on Σ, let $\preccurlyeq^*$ be the ordering on Σ^* defined as follows: for $u, v \in \Sigma^*$, we have $u \preccurlyeq^* v$ if $u = a_1 \cdots a_n$ with $a_i \in \Sigma$, $v = v_0 b_1 v_1 \cdots v_{n-1} b_n v_n$, with $v_i \in \Sigma^*$, $b_j \in \Sigma$, and for all $i = 1, \ldots, n$, we have $a_i \preccurlyeq b_i$. In other words, u is obtained from v by removing some letters, and then replacing some of the remaining letters by smaller ones. Higman's Lemma is the following result, see [4] for instance for a proof.

▶ **Lemma 8.** ***(Higman)*** *If $\preccurlyeq$ is a well ordering on A, then $\preccurlyeq^*$ is a well ordering on A^*.*

We extend the multiplication on $\mathbb{N}_\omega$ by $\omega \cdot 0 = 0 = 0 \cdot \omega$ and $\omega \cdot k = \omega = k \cdot \omega$ if $k \neq 0$. This multiplication then extends componentwise to the scalar multiplication of $\mathbb{N}_\omega^d$ by $\mathbb{N}_\omega$.

▶ **Proposition 9.** *Let $\mathcal{S} = \langle A, \delta, \boldsymbol{x}_{in} \rangle$ be a VAS. Then*

$$\mathsf{Lim\,Reach}(\mathcal{S}) = \left\{ \boldsymbol{x}_{in} + \delta(v) + \omega\delta(\pi) \mid \pi \text{ is productive for } v \right\}.$$

Proof. For the inclusion from right to left, if π is a productive sequence for a word v, then $\boldsymbol{x}_{\text{in}} + \delta(v) + \omega\delta(\pi)$ is the limit of the sequence $(\boldsymbol{x_n})_{n\in\mathbb{N}}$ with $\boldsymbol{x_n} = \boldsymbol{x}_{\text{in}} + \delta(v) + n\delta(\pi)$, and $\boldsymbol{x_n}$ is a reachable state by Lemma 7. We prove the reverse inclusion thanks to Higman's lemma.

We first introduce a well ordering $\sqsubseteq$ over $\mathsf{Reach}(\mathcal{S})$, using a temporary ordering $\preccurlyeq$. Consider the infinite set $\Sigma = A \times \mathbb{N}_\omega^d$. This set is well ordered by $\preccurlyeq$, defined by

$$(a, \boldsymbol{y}) \preccurlyeq (b, \boldsymbol{z}) \text{ if and only if } a = b \text{ and } \boldsymbol{y} \leqslant \boldsymbol{z}.$$

Since $\preccurlyeq$ is a well ordering, Higman's lemma shows that $\preccurlyeq^*$ is a well-ordering over Σ^*. Let us now associate to every reachable state $\boldsymbol{y} \in \mathsf{Reach}(\mathcal{S})$ a word $\alpha_{\boldsymbol{y}}$ in Σ^* as follows: since $\boldsymbol{y}$ is reachable, we can choose a word $v = a_1 \cdots a_k$, with $a_i \in A$, such that $\boldsymbol{x}_{\text{in}} \xrightarrow{v} \boldsymbol{y}$. We introduce the sequence $(\boldsymbol{y_i})_{0\leqslant i\leqslant k}$ of states defined by $\boldsymbol{y_i} = \boldsymbol{x}_{\text{in}} + \delta(a_1 \cdots a_i)$, and we let:

$$\alpha_{\boldsymbol{y}} = (a_1, \boldsymbol{y_1}) \cdots (a_k, \boldsymbol{y_k}).$$

The ordering $\sqsubseteq$ over $\mathsf{Reach}(\mathcal{S})$ is defined by $\boldsymbol{y} \sqsubseteq \boldsymbol{z}$ if $\alpha_{\boldsymbol{y}} \preccurlyeq^* \alpha_{\boldsymbol{z}}$ and $\boldsymbol{y} \leqslant \boldsymbol{z}$. Since the orderings $\preccurlyeq^*$ over Σ^* and $\leqslant$ over $\mathbb{N}^d$ are well, we deduce that $\sqsubseteq$ is a well ordering over $\mathsf{Reach}(\mathcal{S})$.

To show the inclusion from left to right, pick $\boldsymbol{x} \in \mathsf{Lim\,Reach}(\mathcal{S})$: $\boldsymbol{x}$ is the limit of a sequence $(\boldsymbol{x_k})_{k\in\mathbb{N}}$ of reachable states. By extracting a subsequence we can assume that $(\boldsymbol{x_k}(i))_{k\in\mathbb{N}}$ is strictly increasing if $\boldsymbol{x}(i) = \omega$, and $\boldsymbol{x_k}(i) = \boldsymbol{x}(i)$ if $\boldsymbol{x}(i) < \omega$. Denote by α_k the word $\alpha_{\boldsymbol{x_k}}$ associated to the reachable state $\boldsymbol{x_k}$. Since $\sqsubseteq$ is a well ordering, there exist $m < n$ such that $\boldsymbol{x_m} \sqsubseteq \boldsymbol{x_n}$. By construction of α_m there exists a word $v = a_1 \cdots a_k$ with $a_j \in A$ such that the sequence $(\boldsymbol{y_j})_{0\leqslant j\leqslant k}$ defined by $\boldsymbol{y_j} = \boldsymbol{x}_{\text{in}} + \delta(a_1 \cdots a_j)$ for every $j \in \{1, \ldots, k\}$ satisfies:

$$\alpha_m = (a_1, \boldsymbol{y_1}) \cdots (a_k, \boldsymbol{y_k}).$$

Since $\boldsymbol{x_m} \preccurlyeq^* \boldsymbol{x_n}$ and by definition of $\preccurlyeq^*$, there exist a sequence $(\boldsymbol{z_j})_{1\leqslant j\leqslant k}$ of states with $\boldsymbol{y_j} \leqslant \boldsymbol{z_j}$, and a sequence $(\beta_j)_{0\leqslant j\leqslant k}$ of words in Σ^* such that the following equality holds:

$$\alpha_n = \beta_0(a_1, \boldsymbol{z_1})\beta_1 \cdots (a_k, \boldsymbol{z_k})\beta_k$$

We call *label* of a word $(b_1, \boldsymbol{t_1}) \cdots (b_\ell, \boldsymbol{t_\ell})$ over Σ the word $b_1 \cdots b_\ell$ over A. Consider the sequence $\pi = (u_j)_{0\leqslant j\leqslant k}$ where u_j is the label of β_j. By definition of α_n, we have

$$\boldsymbol{x}_{\text{in}} \xrightarrow{u_0 a_1} \boldsymbol{z_1} \cdots \xrightarrow{u_{k-1} a_k} \boldsymbol{z_k} \xrightarrow{u_k} \boldsymbol{x_n}$$

In particular, $\boldsymbol{z_j} = \boldsymbol{y_j} + \delta(u_0) + \cdots + \delta(u_{j-1})$ for every $j \in \{1, \ldots, k\}$ and $\boldsymbol{x_n} = \boldsymbol{z_k} + \delta(u_k) = \boldsymbol{y_k} + \delta(\pi) = \boldsymbol{x_m} + \delta(\pi)$. As $\boldsymbol{y_j} \leqslant \boldsymbol{z_j}$ for every $j \in \{1, \ldots, k\}$ and $\boldsymbol{x_m} \leqslant \boldsymbol{x_n}$, we deduce that π is productive for v.

Finally, let us prove that $\boldsymbol{x} = \boldsymbol{y}$ where $\boldsymbol{y} = \boldsymbol{x}_{\text{in}} + \delta(v) + \omega\delta(\pi)$. We have $\boldsymbol{x_n} = \boldsymbol{x_m} + \delta(\pi)$. Let us consider $i \in \{1, \ldots, d\}$. If $\boldsymbol{x}(i) < \omega$ then $\boldsymbol{x_m}(i) = \boldsymbol{x}(i) = \boldsymbol{x_n}(i)$. Thus $\delta(\pi)(i) = 0$ and we deduce that $\boldsymbol{x}(i) = \boldsymbol{y}(i)$. If $\boldsymbol{x}(i) = \omega$ then $\boldsymbol{x_m}(i) < \boldsymbol{x_n}(i)$ and we deduce that $\delta(\pi)(i) > 0$ and in particular $\boldsymbol{x}(i) = \omega = \boldsymbol{y}(i)$. Thus $\boldsymbol{x} = \boldsymbol{y}$. We have proved that there exists a productive sequence π for a word v such that $\boldsymbol{x} = \boldsymbol{x}_{\text{in}} + \delta(v) + \omega\delta(\pi)$. ◀

It is easier to prove that the *complement* of $\mathsf{Lim\,Reach}(\mathcal{S})$ recursively enumerable. We just give the construction. Let $\mathcal{S} = \langle A, \delta, \boldsymbol{x}_{\text{in}} \rangle$ and $\boldsymbol{y} \in \mathbb{N}_\omega^d$. We introduce d distinct additional elements $b_1, \ldots, b_d \notin A$. Let $B = \{b_1, \ldots, b_d\}$. We introduce the VAS $\mathcal{S}_{\boldsymbol{y}} = \langle A \uplus B, \delta_{\boldsymbol{y}}, \boldsymbol{x}_{\text{in}} \rangle$, where $\delta_{\boldsymbol{y}}$ extends δ by:

$$\delta_{\boldsymbol{y}}(b_i) = \begin{cases} \boldsymbol{0} & \text{if } \boldsymbol{y}(i) < \omega, \\ -\boldsymbol{e_i} & \text{if } \boldsymbol{y}(i) = \omega. \end{cases}$$

Finally, we define from $\boldsymbol{y}$ a sequence $(\boldsymbol{y_\ell})_\ell$ converging to $\boldsymbol{y}$, by $\boldsymbol{y_\ell}(i) = \begin{cases} \ell & \text{if } \boldsymbol{y}(i) = \omega, \\ \boldsymbol{y}(i) & \text{if } \boldsymbol{y}(i) < \omega. \end{cases}$

▶ **Lemma 10.** *Let $\mathcal{S}_{\boldsymbol{y}}$ and $(\boldsymbol{y_\ell})_\ell$ constructed from $\boldsymbol{y}$ as above. Then,*

$$\boldsymbol{y} \notin \mathsf{Lim\,Reach}(\mathcal{S}) \iff \exists \ell \in \mathbb{N},\ \boldsymbol{y_\ell} \notin \mathsf{Reach}(\mathcal{S}_{\boldsymbol{y}}). \tag{1}$$

In particular, the complement of $\mathsf{Lim\,Reach}(\mathcal{S})$ is recursively enumerable.

Theorem 5 now follows from Proposition 9 and Lemma 10.

4 Between the cover and the reachability set: the filtered covers

In this section, we introduce a set hybrid between the reachability and cover sets, which to our knowledge has not yet been considered. Instead of the downward closure $\mathsf{Cover}(\mathcal{S})$ of $\mathsf{Reach}(\mathcal{S})$ wrt. the pointwise ordering $\leqslant$, we consider $\mathsf{Cover}_{\leqslant_P}(\mathcal{S}) = {\downarrow_{\leqslant_P}}\mathsf{Reach}(\mathcal{S})$, that is, we replace $\leqslant$ with an ordering $\leqslant_P$ parametrized by a set of "positions" $P \subseteq \{1, \ldots, d\}$:

$$\boldsymbol{x} \leqslant_P \boldsymbol{y} \quad \text{if} \quad \begin{cases} \boldsymbol{x}(i) = \boldsymbol{y}(i) & \text{for } i \in P, \\ \boldsymbol{x}(i) \leqslant \boldsymbol{y}(i) & \text{for } i \notin P. \end{cases}$$

The set P contains the components for which we insist on keeping equality. Thus, $\leqslant_\emptyset$ is the usual pointwise ordering $\leqslant$, while $\leqslant_{\{1,\ldots,d\}}$ boils down to equality. Note that $\leqslant_P$ is *not* a well ordering, except if $P = \emptyset$ (*e.g.*, $\mathbb{N}$ ordered by $\leqslant_{\{1\}}$ consists only of incomparable elements).

The ordering $\leqslant_{\{1\}}$ will be abbreviated as $\leqslant_1$. It is a natural order to study for a VAS_0 (recall that the zero-test occurs on the first component). Indeed, the transition relation of a VAS_0 is monotonic regarding this order: if $\boldsymbol{x} \xrightarrow{u} \boldsymbol{x'}$ and $x \leqslant_1 y$, then there exists $\boldsymbol{y'}$ with $\boldsymbol{y} \xrightarrow{u} \boldsymbol{y'}$ and $\boldsymbol{x'} \leqslant_1 \boldsymbol{y'}$. More precisely, testing if $\mathsf{Cover}_{\leqslant_1}(\mathcal{S})$ contains a vector whose first component is 0 is what we need to design our algorithm computing the cover of a VAS with one zero test. Unfortunately, this set has infinitely many maximal elements for $\leqslant_1$, and thus cannot be represented by a finite basis. The following theorem shows that we cannot find a sensible way to compute a representation of this set, as any representation would not allow to test for equality.

▶ **Theorem 11.** *Given two VAS $\mathcal{S}_1$ and $\mathcal{S}_2$ of the same dimension d, the equality problem $\mathsf{Cover}_{\leqslant_1}(\mathcal{S}_1) = \mathsf{Cover}_{\leqslant_1}(\mathcal{S}_2)$ is undecidable.*

Proof. We reduce this problem to the equality problem $\mathsf{Reach}(\mathcal{S}_1) = \mathsf{Reach}(\mathcal{S}_2)$. This problem is known to be undecidable [12]. Let us first consider a VAS $\mathcal{S} = \langle A, \delta, \boldsymbol{x}_{\text{in}} \rangle$ of dimension d. We introduce a VAS $\mathcal{S}' = \langle A, \delta', \boldsymbol{x}'_{\text{in}} \rangle$ of dimension $d+1$ that counts in the first component the sum of the other components. Formally, $\boldsymbol{x}'_{\text{in}} = (\sum_{i=1}^{d} \boldsymbol{x}_{\text{in}}(i), \boldsymbol{x}_{\text{in}})$ and $\delta'(a) = (\sum_{i=1}^{d} \delta(a)(i), \delta(a))$ for every $a \in A$. Observe that the following equivalence holds:

$$(n, \boldsymbol{x}) \in \mathsf{Reach}(\mathcal{S}') \iff \boldsymbol{x} \in \mathsf{Reach}(\mathcal{S}) \text{ and } n = \sum_{i=1}^{d} \boldsymbol{x}(i)$$

Finally let us consider two VAS $\mathcal{S}_1$ and $\mathcal{S}_2$ and just observe that $\mathsf{Reach}(\mathcal{S}_1) = \mathsf{Reach}(\mathcal{S}_2)$ if and only if $\mathsf{Cover}_{\leqslant_1}(\mathcal{S}'_1) = \mathsf{Cover}_{\leqslant_1}(\mathcal{S}'_2)$. ◀

So, we cannot hope for a useful representation of the sets $\mathsf{Cover}_{\leqslant_P}(\mathcal{S})$ themselves. However, one can capture the needed information differently, by replacing the downward closure $\downarrow_{\leqslant_P}$ in $\mathsf{Cover}_{\leqslant_P}(\mathcal{S}) = \downarrow_{\leqslant_P} \mathsf{Reach}(\mathcal{S})$ with an operator $\Downarrow_{\boldsymbol{f}}$ parametrized by a vector $\boldsymbol{f}$ of $\mathbb{N}^d_\omega$. Informally, $\Downarrow_{\boldsymbol{f}} M$ takes into account only elements of M that agree with $\boldsymbol{f}$ on its finite components. Formally, for $\boldsymbol{f} \in \mathbb{N}^d_\omega$ and $M \subseteq \mathbb{N}^d$, let

$$\mathsf{Filter}(M, \boldsymbol{f}) = \Big\{ \boldsymbol{x} \in M \mid \bigwedge_{i=1}^{d} \big[\boldsymbol{f}(i) < \omega \implies \boldsymbol{x}(i) = \boldsymbol{f}(i) \big] \Big\},$$
$$\Downarrow_{\boldsymbol{f}} M = \Big\downarrow \mathsf{Filter}(M, \boldsymbol{f}).$$

Note that $\Downarrow_{\boldsymbol{f}} M = \downarrow M$ for $\boldsymbol{f} = (\omega, \omega, \ldots, \omega)$. On the other hand, if $\boldsymbol{f} \in \mathbb{N}^d$, then $\Downarrow_{\boldsymbol{f}} M = \downarrow \boldsymbol{f}$ if $\boldsymbol{f} \in M$, and $\Downarrow_{\boldsymbol{f}} M = \emptyset$ otherwise. Observe also that $\Downarrow_{\boldsymbol{f}} M$ is downward closed and that the maximal elements of any basis of $\Downarrow_{\boldsymbol{f}} M$ agree with $\boldsymbol{f}$ on every component i where $\boldsymbol{f}(i)$ is finite. The next lemma provides a relationship between the sets $\Downarrow_{\boldsymbol{f}} M$ and $\downarrow_{\leqslant_P} M$.

▶ **Lemma 12.** *Let $M \subseteq \mathbb{N}^d$. Then, the following conditions are equivalent:*

(*a*) *For all $\boldsymbol{f} \in \mathbb{N}^d_\omega$, one can effectively compute the basis of $\Downarrow_{\boldsymbol{f}} M$.*
(*b*) *For all $P \subseteq \{1, \ldots, d\}$, the set $\mathsf{Lim} \downarrow_{\leqslant_P} M$ is recursive.*

The main result of this section states that both conditions of Lemma 12 actually hold when M is the reachability set of a VAS. This is obtained by first proving that $\mathsf{Cover}_{\leqslant_P}(\mathcal{S}) = \mathsf{Reach}(\mathcal{S}_P)$ where $\mathcal{S}_P$ is a VAS constructed from $\mathcal{S}$ and P. From this equality, we deduce that $\mathsf{Lim}\,\mathsf{Cover}_{\leqslant_P}(\mathcal{S}) = \mathsf{Lim}\,\mathsf{Reach}(\mathcal{S}_P)$. Applying Theorem 5, it follows that this set is recursive, which proves condition (*b*) for $M = \mathsf{Reach}(\mathcal{S})$. Then by Lemma 12, condition (*a*) also holds.

Let $\mathcal{S} = \langle A, \delta, \boldsymbol{x}_{\text{in}} \rangle$ be a VAS and $P \subseteq \{1, \ldots, d\}$. Let us define a VAS $\mathcal{S}_P$ such that $\mathsf{Reach}(\mathcal{S}_P) = \mathsf{Cover}_{\leqslant_P}(\mathcal{S})$. We consider d distinct additional elements $b_1, \ldots, b_d \notin A$. Let $B = \{b_1, \ldots, b_d\}$. We consider the VAS $\mathcal{S}_P = \langle A \uplus B, \delta_P, \boldsymbol{x}_{\text{in}} \rangle$, where δ_P extends δ by:

$$\delta_P(b_i) = \begin{cases} \mathbf{0} & \text{if } i \in P \\ -\boldsymbol{e_i} & \text{if } i \notin P. \end{cases}$$

▶ **Lemma 13.** *Let $\mathcal{S}_P$ constructed from $\mathcal{S}$ and P as above. Then $\mathsf{Cover}_{\leqslant_P}(\mathcal{S}) = \mathsf{Reach}(\mathcal{S}_P)$.*

Proof. Consider a state $\boldsymbol{x} \in \mathsf{Cover}_{\leqslant_P}(\mathcal{S})$. By definition, there exists $\boldsymbol{y} \in \mathsf{Reach}(\mathcal{S})$ such that $\boldsymbol{x} \leqslant_P \boldsymbol{y}$. Observe that $\boldsymbol{x}_{\text{in}} \xrightarrow{*} \boldsymbol{y} \xrightarrow{u} \boldsymbol{x}$ in $\mathcal{S}_P$ with $u = \prod_{i=1}^{d} b_i^{\boldsymbol{y}(i) - \boldsymbol{x}(i)}$. Hence $\boldsymbol{x} \in \mathsf{Reach}(\mathcal{S}_P)$. Conversely let $\boldsymbol{x} \in \mathsf{Reach}(\mathcal{S}_P)$ and let $u \in (A \cup B)^*$ such that $\boldsymbol{x}_{\text{in}} \xrightarrow{u} \boldsymbol{x}$ in $\mathcal{S}_P$. Consider the word v obtained from u by erasing all letters of B. Since $\delta_P(b) \leqslant \mathbf{0}$ for $b \in B$, the word v is still fireable from $\boldsymbol{x}_{\text{in}}$. Thus $\boldsymbol{y} = \boldsymbol{x}_{\text{in}} + \delta(v) \in \mathsf{Reach}(\mathcal{S})$. Moreover, by definition of $\mathcal{S}_P$ we have $\boldsymbol{x} \leqslant_P \boldsymbol{y}$. Therefore $\boldsymbol{x} \in \mathsf{Cover}_{\leqslant_P}(\mathcal{S})$. ◀

Combining Lemma 13, Theorem 5 and Lemma 12 as explained above yields:

▶ **Theorem 14.** *Given $\boldsymbol{f} \in \mathbb{N}_\omega^d$ and a VAS $\mathcal{S}$, one can effectively compute a basis of $\Downarrow_{\boldsymbol{f}}\mathsf{Reach}(\mathcal{S})$.*

5 Computing the cover of a VAS with one zero-test

We provide an algorithm computing the basis of $\mathsf{Cover}(\mathcal{S})$ of any VAS_0 $\mathcal{S} = \langle A, a_Z, \delta, \boldsymbol{x}_{\mathrm{in}}\rangle$. Intuitively the algorithm, inspired by the Karp and Miller algorithm for VAS [15], builds a tree with nodes labeled by vectors in $\{0\} \times \mathbb{N}_\omega^{d-1}$ such that the finite set R of node labels satisfies the following equality when the algorithm terminates:

$$\Downarrow_{\boldsymbol{f}}\mathsf{Reach}(\mathcal{S}) = (\downarrow R) \cap \mathbb{N}^d, \quad \text{where } \boldsymbol{f} = (0, \omega, \ldots, \omega).$$

In order to simplify the presentation, we assume without loss of generality that $\boldsymbol{x}_{\mathrm{in}} \in \{0\} \times \mathbb{N}^{d-1}$ and $\delta(a_Z) \in \{0\} \times \mathbb{Z}^{d-1}$. In the sequel we denote by $\mathcal{S}_{\mathrm{VAS}}$ the VAS $\mathcal{S}_{\mathrm{VAS}} = (A, \delta, \boldsymbol{x}_{\mathrm{in}})$ obtained from $\mathcal{S}$ by removing the zero-test a_Z. Moreover, given $\boldsymbol{s} \in \{0\} \times \mathbb{N}^{d-1}$ we denote by $\mathcal{S}(\boldsymbol{s})$ and $\mathcal{S}_{\mathrm{VAS}}(\boldsymbol{s})$ the VASs obtained respectively from $\mathcal{S}$ and $\mathcal{S}_{\mathrm{VAS}}$ by replacing the initial state $\boldsymbol{x}_{\mathrm{in}}$ by $\boldsymbol{s}$.

At any step of the execution, in the tree built in the algorithm, every ancestor node n of a node n' satisfies the invariant $\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{x'}$ where $\boldsymbol{x}, \boldsymbol{x'}$ are the labels of n, n' and where $\overset{*}{\Rightarrow}$ is the binary relation defined over the vectors in $\{0\} \times \mathbb{N}_\omega^{d-1}$ by:

$$\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{x'} \text{ if } (\downarrow \boldsymbol{x'}) \cap \mathbb{N}^d \subseteq \bigcup_{\boldsymbol{s} \in (\downarrow \boldsymbol{x}) \cap \mathbb{N}^d} \Downarrow_{\boldsymbol{f}}\mathsf{Reach}(\mathcal{S}(\boldsymbol{s})).$$

By the next lemma, it is sufficient to maintain this invariant along each parent-child edge.

▶ **Lemma 15.** *The binary relation $\overset{*}{\Rightarrow}$ is reflexive and transitive.*

Proof. The reflexivity is immediate. For the transitivity, we first introduce the binary relation $\xrightarrow{*}$ over $\mathbb{N}^d$ defined by $\boldsymbol{x} \xrightarrow{*} \boldsymbol{x'}$ if there exists $u \in (A \cup \{a_Z\})^*$ such that $\boldsymbol{x} \xrightarrow{u} \boldsymbol{x'}$. We observe that $\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{x'}$ if and only if the following relation holds:

$$\forall \boldsymbol{s'} \in (\downarrow \boldsymbol{x'}) \cap \mathbb{N}^d \; \exists \boldsymbol{s} \in (\downarrow \boldsymbol{x}) \cap \mathbb{N}^d \; \exists \boldsymbol{z} \in \{0\} \times \mathbb{N}_\omega^{d-1} \quad \boldsymbol{s} \xrightarrow{*} \boldsymbol{s'} + \boldsymbol{z}.$$

Assume that $\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{x'}$ and $\boldsymbol{x'} \overset{*}{\Rightarrow} \boldsymbol{x''}$. Let $\boldsymbol{s''} \in (\downarrow \boldsymbol{x''}) \cap \mathbb{N}^d$. From $\boldsymbol{x'} \overset{*}{\Rightarrow} \boldsymbol{x''}$, we deduce that there exist $\boldsymbol{z'} \in \{0\} \times \mathbb{N}^{d-1}$ and $\boldsymbol{s'} \in (\downarrow \boldsymbol{x'}) \cap \mathbb{N}^d$ such that $\boldsymbol{s'} \xrightarrow{*} \boldsymbol{s''} + \boldsymbol{z'}$. From $\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{x'}$, we deduce that there exist $\boldsymbol{z} \in \{0\} \times \mathbb{N}^{d-1}$ and $\boldsymbol{s} \in (\downarrow \boldsymbol{x}) \cap \mathbb{N}^d$ such that $\boldsymbol{s} \xrightarrow{*} \boldsymbol{s'} + \boldsymbol{z}$. In particular we deduce that $\boldsymbol{s} \xrightarrow{*} \boldsymbol{s''} + \boldsymbol{z} + \boldsymbol{z'}$. We have proved that $\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{x''}$. ◀

Assume now that $\boldsymbol{x} \in \{0\} \times \mathbb{N}_\omega^{d-1}$ labels a leaf. We create a child of this leaf if the vector $\boldsymbol{y} = \boldsymbol{x} + \delta(a_Z)$ is nonnegative. Note that in this case $\boldsymbol{y} \in \{0\} \times \mathbb{N}_\omega^{d-1}$, since $\delta(a_Z)(1) = 0$. We do not violate the invariant when creating the child labeled $\boldsymbol{y}$ since $\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{y}$. We may also add new children labeled by elements of the minimal basis $B(\boldsymbol{x})$ of the following downward-closed set:

$$\bigcup_{\boldsymbol{s} \in (\downarrow \boldsymbol{x}) \cap \mathbb{N}^d} \Downarrow_{\boldsymbol{f}}\mathsf{Reach}(\mathcal{S}_{\mathrm{VAS}}(\boldsymbol{s}))$$

We observe that $\boldsymbol{x} \overset{*}{\Rightarrow} \boldsymbol{b}$ for every $\boldsymbol{b} \in B(\boldsymbol{x})$, so that the invariant will still be fulfilled after adding elements of $B(\boldsymbol{x})$.

▶ **Lemma 16.** *The basis $B(\boldsymbol{x})$ is effectively computable.*

Proof. We introduce the set I of components $i \in \{2, \dots, d\}$ such that $\boldsymbol{x}(i) = \omega$. We consider the VAS $\mathcal{S}'_{\mathrm{VAS}} = (A, \delta', \boldsymbol{x}')$ obtained from $\mathcal{S}_{\mathrm{VAS}}(\boldsymbol{x})$ by preventing any modification of components in I. More formaly δ' and $\boldsymbol{x}'$ are defined by $\delta'(a)(i) = 0$ and $\boldsymbol{x}'(i) = 0$ if $i \in I$ and $\delta'(a)(i) = \delta(a)(i)$ and $\boldsymbol{x}'(i) = \boldsymbol{x}(i)$ if $i \notin I$. Theorem 14 shows that we can effectively compute the basis B' of $\Downarrow_{\boldsymbol{f}}\mathsf{Reach}(\mathcal{S}'_{\mathrm{VAS}})$. Now $B(\boldsymbol{x}) = \{y + z \mid y \in B'\}$, where $\boldsymbol{z}$ is the vector defined by $\boldsymbol{z}(i) = \omega$ if $i \in I$ and $\boldsymbol{z}(i) = 0$ if $i \notin I$. ◀

The algorithm termination is obtained by introducing an *acceleration operator* ∇. We define the vector $\boldsymbol{x} \nabla \boldsymbol{y}$ for every $\boldsymbol{x}, \boldsymbol{y} \in \{0\} \times \mathbb{N}_\omega^{d-1}$ such that $\boldsymbol{x} \leqslant \boldsymbol{y}$ by

$$(\boldsymbol{x} \nabla \boldsymbol{y})(i) = \begin{cases} \omega & \text{if } \boldsymbol{x}(i) < \boldsymbol{y}(i) \\ \boldsymbol{x}(i) & \text{if } \boldsymbol{x}(i) = \boldsymbol{y}(i). \end{cases}$$

▶ **Lemma 17.** *If $\boldsymbol{x} \stackrel{*}{\Rightarrow} \boldsymbol{y}$ with $\boldsymbol{x} \leqslant \boldsymbol{y}$ then $\boldsymbol{x} \stackrel{*}{\Rightarrow} \boldsymbol{x} \nabla \boldsymbol{y}$.*

Let us now describe informally the algorithm. It inductively computes a tree with nodes labeled by vectors in $\{0\} \times \mathbb{N}_\omega^{d-1}$. The tree is rooted at a node labeled by $\boldsymbol{x}_{\mathrm{in}}$ (recall that $\boldsymbol{x}_{\mathrm{in}} \in \{0\} \times \mathbb{N}^{d-1}$). The tree is modified in such a way that for every node n and for every child n' of n, the labels $\boldsymbol{x}, \boldsymbol{x'}$ of n, n' satisfy $\boldsymbol{x} \stackrel{*}{\Rightarrow} \boldsymbol{x'}$. While there exists a leaf n' labeled by a vector $\boldsymbol{x'}$ that admits an ancestor n labeled by a vector $\boldsymbol{x}$ such that $\boldsymbol{x} \leqslant \boldsymbol{x'} < \boldsymbol{x} \nabla \boldsymbol{x'}$, we replace the label $\boldsymbol{x'}$ of node n' by $\boldsymbol{x} \nabla \boldsymbol{x'}$. From Lemma 17, we deduce that the invariant still holds. Since this loop just replaces some components by ω, it terminates. Then, the algorithm checks if for every leaf n labeled by $\boldsymbol{x}$, there exists a strict ancestor (*i.e.*, different from n) labeled by the same vector $\boldsymbol{x}$. In this case, the algorithm terminates and it returns the set of node labels. Otherwise the algorithm considers a leaf n not fulfilling this condition, and it creates a new child of n labeled by $\boldsymbol{b}$ for each $\boldsymbol{b} \in B(\boldsymbol{x})$. It also creates a new child labeled by $\boldsymbol{x} + \delta(a_Z)$ if this vector is nonnegative. The modification of the tree is then restarted.

The termination of this algorithm follows from König's lemma. If the algorithm does not terminate, then it would generate an infinite tree. Because this tree has a finite branching degree, by König's lemma, there is an infinite branch. Since $\leqslant$ is a well-ordering over $\{0\} \times \mathbb{N}_\omega^{d-1}$, this implies that we can extract from this infinite branch an infinite increasing subsequence. However, since we add children to a leaf only if there does not exist a strict ancestor labeled by the same vector, this sequence cannot contain the same vector twice, and must therefore be *strictly* increasing. But, due to the use of the operator ∇, a component with an integer is replaced by ω at every acceleration step. Because the number of ω's in the vectors labeling a branch cannot decrease, we obtain a contradiction. We deduce the following proposition.

▶ **Proposition 18.** *Algorithm 1 terminates and it returns a finite set R such that*

$$\Downarrow_{\boldsymbol{f}}\mathsf{Reach}(\mathcal{S}) = \downarrow R \cap \mathbb{N}^d.$$

We have proved that we can effectively compute a basis R of $\Downarrow_{\boldsymbol{f}}\mathsf{Reach}(\mathcal{S})$. Now, observe that the following equality holds:

$$\mathsf{Cover}(\mathcal{S}) = \bigcup_{\boldsymbol{b} \in R} \bigcup_{\boldsymbol{s} \in (\downarrow \boldsymbol{b}) \cap \mathbb{N}^d} \mathsf{Cover}(\mathcal{S}_{\mathrm{VAS}}(\boldsymbol{s})).$$

Algorithm 1 An algorithm to compute a basis of $\Downarrow_f \mathsf{Reach}(\mathcal{S})$

- Inputs: A VAS_0 $\mathcal{S}$ such that $\boldsymbol{x}_{\mathrm{in}} \in \{0\} \times \mathbb{N}^{d-1}$ and a $\delta(a_Z) \in \{0\} \times \mathbb{Z}^{d-1}$.
- Outputs: R, a finite subset of $\{0\} \times \mathbb{N}_\omega^{d-1}$.
- Internal Variables:
  - $\mathcal{T}$, a tree labeled by elements of $\mathbb{N}_\omega^d$.
  - N, a set of nodes.
- Algorithm:

1: Initialize $\mathcal{T}$ as a single root n_{in}, labeled by $\boldsymbol{x}_{\mathrm{in}}$
2: $N \leftarrow \{n_{in}\}$
3: **while** $N \neq \emptyset$ **do**
4: Take a node n from N
5: $\boldsymbol{x} \leftarrow \mathit{label}(n)$
6: **if** the label of every strict ancestor of n is not equal to $\boldsymbol{x}$ **then**
7: **for all** strict ancestor n_0 of n **do**
8: $\boldsymbol{x}_0 \leftarrow \mathit{label}(n_0)$
9: **if** $\boldsymbol{x}_0 \leqslant \boldsymbol{x}$ **then**
10: $\boldsymbol{x} \leftarrow \boldsymbol{x}_0 \nabla \boldsymbol{x}$
11: **end if**
12: **end for**
13: Replace the label of n by $\boldsymbol{x}$
14: **if** $\boldsymbol{x} + \delta(a_Z) \geqslant \boldsymbol{0}$ **then**
15: Create a new node in $\mathcal{T}$ labeled by $\boldsymbol{x} + \delta(a_Z)$, as a child of n
16: Add this node to N
17: **end if**
18: **for all** $\boldsymbol{b} \in B(\boldsymbol{x})$ **do**
19: Create a new node in $\mathcal{T}$ labeled by $\boldsymbol{b}$, as a child of n
20: Add this node to N
21: **end for**
22: **end if**
23: **end while**
24: $R \leftarrow \{\mathit{label}(n) \mid n \in \mathit{nodes}(\mathcal{T})\}$
25: **return** R

A reduction similar to the one provided in the proof of Lemma 16 shows that the basis of $\bigcup_{\boldsymbol{s}\in(\downarrow\boldsymbol{b})\cap\mathbb{N}^d} \mathsf{Cover}(\mathcal{S}_{\mathrm{VAS}}(\boldsymbol{s}))$ can be obtained from a basis of $\mathsf{Cover}(\mathcal{S}'_{\mathrm{VAS}})$, where $\mathcal{S}'_{\mathrm{VAS}}$ is a VAS obtained from $\mathcal{S}_{\mathrm{VAS}}$ and $\boldsymbol{b}$ by removing the components $i \in \{2,\dots,d\}$ such that $\boldsymbol{b}(i) = \omega$. We deduce the following theorem.

▶ **Theorem 19.** *Given a VAS$_0$ $\mathcal{S}$, one can effectively compute the finite basis of* $\mathsf{Cover}(\mathcal{S})$.

6 Conclusion and perspectives

Our main result is a forward algorithm, à la Karp&Miller, to compute the downward closure of the reachability set of a nonmonotonic transition system: VAS$_0$. This implies that place-boundedness is decidable. For our purposes, we introduced new sets, sitting between the cover and the reachability set. Unfortunately, we cannot say anything about the complexity of the computation of the cover for VAS$_0$, because our proof uses the decidability of the reachability problem for VAS as an oracle, whose complexity is still open.

Since we have solved the place-boundedness problem, a natural question would be an instance of a liveness problem, like the repeated control-state reachability problem (RCSRP). One could think of a reduction from the RCSRP to the place-boundedness problem (or to the computation of the cover), by adding a new counter c_q getting increased each time the control-state q is hit. This does actually not work, because c_q might be unbounded even if on each single run, it is bounded. It seems that these two problems are not close: for solving the RCSRP, we need to decide whether there is an infinite run along which a given counter is unbounded, while the cover gives boundedness information about the *global* reachability set, but not on infinite runs. For VAS with one *weak* zero-test (for instance a lossy zero-test, like a reset), the usual Karp and Miller algorithm can be easily extended, and the RCSRP is decidable; for VAS with two weak zero-test (two resets), the techniques used in [6] allow one to show that this problem is undecidable. Finally, the RCSRP remains open for VAS$_0$.

We have proved new decidability results for VAS$_0$. One could think that maybe, VAS$_0$ can be simulated by VAS. The answer is negative: the language $\{a^n b^n \mid n \geqslant 1\}^*$ can be easily recognized by a VAS$_0$, but not by a VAS [16]. More generally, one may prove that for every VAS-language L, there is a VAS$_0$ $\mathcal{S}$ such that $L(\mathcal{S}) = L^*$. One can also separate VAS and VAS$_0$ wrt. the reachability set. Hence, even if their reachability problem is decidable [21] and their cover is computable (this paper), VAS$_0$ are strictly more powerful than VAS.

References

1 P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321, 1996.

2 P. A. Abdulla and R. Mayr. Minimal cost reachability/coverability in priced timed Petri nets. In L. de Alfaro, editor, *FOSSACS*, volume 5504 of *LNCS*, pages 348–363. Springer, 2009.

3 A. Bouajjani and R. Mayr. Model checking lossy vector addition systems. In Ch. Meinel and S. Tison, editors, *STACS'99*, volume 1563, pages 323–333, 1999.

4 R. Diestel. *Graph Theory*, volume 173 of *Graduate Texts in Mathematics*. Springer, fourth edition, 2010.

5 C. Dufourd. *Réseaux de Petri avec Reset/Transfert : décidabilité et indécidabilité*. Thèse de doctorat, Laboratoire Spécification et Vérification, ENS Cachan, France, Oct. 1998.

6 C. Dufourd, P. Jančar, and Ph. Schnoebelen. Boundedness of reset P/T nets. In J. Wiedermann, P. van Emde Boas, and M. Nielsen, editors, *ICALP'99*, volume 1644 of *LNCS*, pages 301–310. Springer, July 1999.

7 A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, part I: Completions. In S. Albers and J.-Y. Marion, editors, *STACS'09*, pages 433–444, 2009.

8 A. Finkel and J. Goubault-Larrecq. Forward analysis for WSTS, Part II: Complete WSTS. In S. Albers et al., editors, *ICALP'09*, volume 5556 of *LNCS*, pages 188–199. Springer, 2009.

9 A. Finkel and A. Sangnier. Mixing coverability and reachability to analyze VASS with one zero-test. In D. Peleg and A. Muscholl, editors, *SOFSEM'10*, volume 5901 of *LNCS*, pages 394–406. Springer, 2010.

10 A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comput. Sci.*, 256(1–2):63–92, 2001.

11 A. Finkel and G. Sutre. Decidability of reachability problems for classes of counters automata. In H. Reichel and S. Tison, editors, *STACS'00*, volume 1770 of *LNCS*, pages 346–357. Springer, 2000.

12 M. Hack. The equality problem for vector addition systems is undecidable. *Theor. Comput. Sci.*, 2(1):77–95, 1976.

13 D. Hauschildt. *Semilinearity of the Reachability Set is Decidable for Petri Nets.* PhD thesis, University of Hamburg, 1990.

14 J. E. Hopcroft and J.-J. Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theor. Comput. Sci.*, 8:135–159, 1979.

15 R. M. Karp and R. E. Miller. Parallel program schemata. *J. Comput. System Sci.*, 2:147–195, 1969.

16 S. R. Kosaraju. Limitations of Dijkstra's semaphore primitives and Petri nets. *SIGOPS Oper. Syst. Rev.*, 7(4):122–126, 1973.

17 S. R. Kosaraju. Decidability of reachability in vector addition systems (preliminary version). In *STOC'82*, pages 267–281. ACM, 1982.

18 J. Leroux. The general vector addition system reachability problem by Presburger inductive invariants. In *LICS'09*, pages 4–13, 2009.

19 E. W. Mayr. An algorithm for the general Petri net reachability problem. In *STOC'81*, pages 238–246. ACM, 1981.

20 R. Mayr. Undecidable problems in unreliable computations. *Theor. Comput. Sci.*, 297(1-3):337–354, 2003.

21 K. Reinhardt. Reachability in Petri Nets with inhibitor arcs. *Electr. Notes Theor. Comput. Sci.*, 223:239–264, 2008.

Model checking time-constrained scenario-based specifications*

S. Akshay[1,2], Paul Gastin[1], Madhavan Mukund[2], and K. Narayan Kumar[2]

1 **LSV, ENS Cachan, INRIA, CNRS, France**
{akshay,Paul.Gastin}@lsv.ens-cachan.fr
2 **Chennai Mathematical Institute, Chennai, India**
{madhavan,kumar}@cmi.ac.in

Abstract

We consider the problem of model checking message-passing systems with real-time requirements. As behavioural specifications, we use message sequence charts (MSCs) annotated with timing constraints. Our system model is a network of communicating finite state machines with local clocks, whose global behaviour can be regarded as a timed automaton. Our goal is to verify that all timed behaviours exhibited by the system conform to the timing constraints imposed by the specification. In general, this corresponds to checking inclusion for timed languages, which is an undecidable problem even for timed regular languages. However, we show that we can translate regular collections of time-constrained MSCs into a special class of event-clock automata that can be determinized and complemented, thus permitting an algorithmic solution to the model checking problem.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.204

1 Introduction

In a distributed system, several agents interact to generate a global behaviour. This interaction is usually specified in terms of scenarios, using message sequence charts (MSCs) [8]. Protocol specifications typically include timing requirements for messages and descriptions of how to recover from timeouts, so a natural and useful extension to MSCs is to add timing constraints between pairs of events, yielding time-constrained MSCs (TCMSCs).

Infinite collections of MSCs are typically described using message sequence graphs (MSGs). An MSG, a finite directed graph with nodes labelled by MSCs, is the most basic form of a High-level Message Sequence Chart (HMSC) [9]. We generalise MSGs to time-constrained MSGs (TCMSGs), where nodes are labelled by TCMSCs and edges may have additional time constraints between nodes.

A natural system model in this setting is a timed message-passing automaton (timed MPA), a set of communicating finite-state machines equipped with clocks that are used to guard transitions, as in timed automata [4]. Just as the runs of timed automata are described in terms of timed words, the interactions exhibited by timed MPAs can be described using timed MSCs—MSCs in which each event is assigned an explicit timestamp. The global state space of a timed MPA defines a timed automaton and in this paper we focus on this simplified global view of timed message-passing systems, though our results go through smoothly for the distributed system model as well.

* Supported by ANR-06-SETI-003 DOTS, ARCUS Île de France-Inde, CMI-TCS Academic Alliance.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 204–215

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Our aim is to check if all timed MSCs accepted by a timed MPA conform to the time constraints given by a TCMSG specification. To make the problem tractable, we focus on *locally synchronized* TCMSGs—those for which the underlying behaviour is guaranteed to be regular [7]. In general, our model checking problem corresponds to checking inclusion for timed languages, which is known to be undecidable even for timed regular languages [2]. Fortunately, it turns out that timing constraints in a TCMSG correspond to a very restricted use of clocks. This allows us to associate with each TCMSG an (extended) event clock automaton that accepts all timed MSCs that are consistent with the timing constraints of the TCMSG. These event clock automata can be determinized and complemented, yielding an algorithmic solution to our model checking problem.

The paper is organized as follows. We begin with some preliminaries where we introduce (timed) MSCs and MSGs and state the model-checking problem. In Section 3 we introduce MSC event clock automata and show that they can be determinized and complemented. The next section has the main technical result: translating locally synchronized TCMSGs to finite state MSC event clock automata, which yields a solution to the model-checking problem in Section 5.

2 Preliminaries

2.1 Message sequence charts

A *message sequence chart (MSC)* describes the messages exchanged between a set *Proc* of processes in a distributed system. The first diagram in Figure 1 is an MSC involving two users and a server. Each process evolves vertically along a lifeline. Messages are shown by arrows between the lifelines of the sender and receiver.

Each message consists of two *events*, send and receive, and is labelled using a finite set of message labels. For instance, the events u_1 and a_1 are the send and receive events of a message labelled *req* from process p (*User1*) to process q (*Server*). Each pair of processes p and q is connected by a dedicated fifo channel (p, q)—for example, the messages sent at s_1 and s_2 are on channel (r, q) and the second message cannot be received before the first one.

Since processes are locally sequential, the set of events E_p along a process p is linearly ordered by a relation denoted $\leq_{pp}$. In addition, for each message sent along a channel (p, q), the send and receive events of the message are related by an ordering relation $\leq_{pq}$. Thus, for example, $a_1 \leq_{qq} a_5$ and $a_3 \leq_{qp} u_2$. Together, the local linear orders $\leq_{pp}$ and the message orders $\leq_{pq}$ generate a partial order $\leq$ over the set of events—for instance, $u_3 \leq s_3$.

Finally, we label each event using a finite alphabet *Act* of communication actions. We write $p!q(m)$ to denote the action where p sends message m to q and $p?q(m)$ to denote the action where p receives message m from q. We abbreviate by $p!q$ and $p?q$ the set of all actions of the form $p!q(m)$ and $p?q(m)$, respectively, over all possible choices of m.

Overall, an MSC can then be captured as a labelled partial order $M = (E, \leq, \lambda)$ where $\lambda : E \to Act$ associates each event with its corresponding action. A *cut* is a subset of events that is downward closed: $c \subseteq E$ is a cut if $\downarrow c = c$, where $\downarrow c = \{e \in E \mid \exists e' \in c.\ e \leq e'\}$.

Like any partial order, an MSC can be reconstructed upto isomorphism from its linearisations, i.e., words over *Act* that extend $\leq$. In fact, the fifo condition on channels ensures that a single linearisation suffices to reconstruct an MSC. In this way, an MSC M corresponds to a set $\mathrm{lin}(M)$ of words over *Act* and a set $\mathcal{L}$ of MSCs defines the word language $\bigcup_{M \in \mathcal{L}} \mathrm{lin}(M)$. We say that a set of MSCs $\mathcal{L}$ is *regular* if its associated word language is regular.

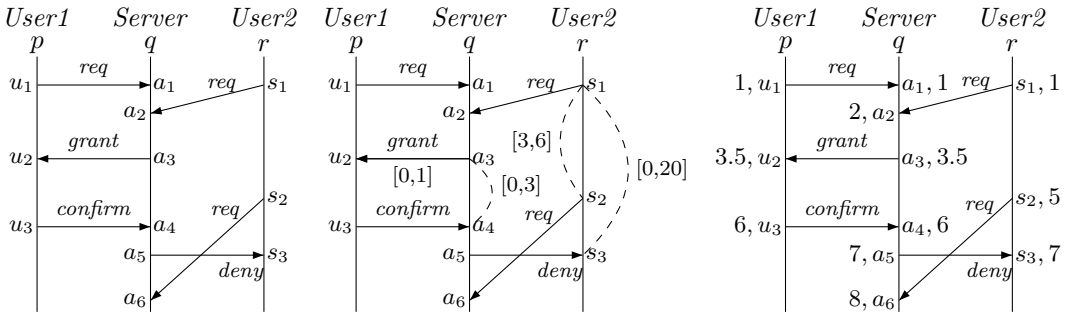


Figure 1 Different views of a system with two users with a server

2.2 Time-constrained message sequence charts

A *time-constrained MSC (TCMSC)* is an MSC annotated with time intervals between pairs of events. We restrict timing constraints to pairs of distinct events on the same process and to the matching send and receive events across messages. Intervals have rational endpoints and may be open or closed at either end.

For example, in the second diagram in Figure 1, the constraint $[0, 3]$ between a_3 and a_4 bounds the time that the *Server* waits for a *User* to confirm a grant. On the other hand, the constraint $[0, 1]$ between a_3 and u_2 bounds the time taken to deliver this particular message.

A TCMSC over Act is a pair $\mathfrak{M} = (M, \tau)$, where $M = (E, \leq, \lambda)$ is an MSC over Act and τ is a partial map from $E \times E$ to the set of intervals such that $(e, e') \in \mathrm{dom}(\tau)$ implies that $e \neq e'$ and either $e \leq_{pp} e'$ or $e \leq_{pq} e'$ for some processes p and q.

2.3 Timed message sequence charts

A *timed MSC (TMSC)* describes a concrete timed behaviour in the MSC setting. In a TMSC, we assign events timestamps that are consistent with the underlying partial order. Thus, a TMSC over Act is a pair $T = (M, t)$ where $M = (E, \leq, \lambda)$ is an MSC over Act and $t : E \to \mathbb{R}_{\geq 0}$ is a function such that if $e \leq e'$ then $t(e) \leq t(e')$ for all $e, e' \in E$.

For instance, consider the TMSC in the third diagram of Figure 1. The message sent at a_3 is received instantaneously while the message sent at s_2 is received 3 time units later.

A timed word over Act is a sequence $(a_1, t_1)(a_2, t_2) \cdots (a_n, t_n)$ where $a_1 a_2 \cdots a_n$ is a word over Act and $t_1 \leq t_2 \leq \cdots \leq t_n$ is a nondecreasing sequence over $\mathbb{R}_{\geq 0}$. The set of timed words over Act is denoted TW_{Act}. A *timed linearisation* of a TMSC is thus a timed word in TW_{Act}. We let t-lin(T) denote the set of timed linearisations of TMSC T. A single TMSC may admit more than one timed linearisation if concurrent events on different processes have the same timestamp. As for untimed MSCs, under the fifo assumption for channels, a timed MSC can be reconstructed from any one of its timed linearisations.

With this definition, TCMSCs can be considered as abstractions of TMSCs and timed words. For instance, we will say that the TMSC in Figure 1 *realises* the TCMSC in the same figure since each interval constraint between events in the TCMSC is satisfied by the time-stamps of the corresponding events in the TMSC. In this way, a TCMSC $\mathfrak{M}$ defines a family of TMSCS—the set of all TMSCs that realise $\mathfrak{M}$, which we denote $\mathcal{L}_{time}(\mathfrak{M})$. We also consider the set $\mathcal{L}_{tw}(\mathfrak{M}) = \bigcup_{T \in \mathcal{L}_{time}(\mathfrak{M})} \text{t-lin}(T)$ of timed words that realise $\mathfrak{M}$.

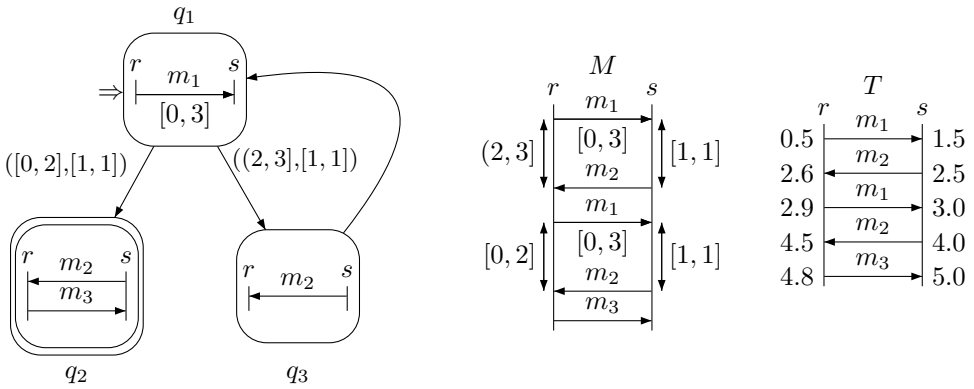


Figure 2 A TCMSG, with a TCMSC and a TMSC that it generates

2.4 Message sequence graphs

A *message sequence graph (MSG)* is a directed graph in which nodes are labelled by MSCs. We begin with a graph $G = (V, \to, v_{in}, V_F)$ with nodes V, initial node $v_{in} \in V$, final nodes $V_F \subseteq V$ and edge relation $\to$. An MSG is a structure $\mathfrak{G} = (G, \mathcal{L}^M, \Phi)$ where $\mathcal{L}^M$ is a set of basic MSCs and $\Phi : V \to \mathcal{L}^M$ associates a basic MSC with each node. An accepting path in G is a sequence of nodes $v_0 v_1 \cdots v_n$ that starts in v_{in} and ends in some node of V_F where each adjacent pair of states is related by $\to$. This path defines an MSC $\Phi(v_0 v_1 \cdots v_n) = \Phi(v_0) \circ \Phi(v_1) \circ \cdots \circ \Phi(v_n)$, where $\circ$ denotes MSC concatenation. When we concatenate two MSCs $M_1 = (E^1, \leq^1, \lambda_1)$ and $M_2 = (E^2, \leq^2, \lambda_2)$ we attach the lifelines in M_2 below those of M_1 to obtain an MSC $M_1 \circ M_2 = (E^1 \cup E^2, \leq, \lambda)$ where λ combines λ_1 and λ_2 and $\leq$ is generated by $\leq^1 \cup \leq^2 \cup \{(e_1, e_2) \mid \exists p.\ e_1 \in E^1_p, e_2 \in E^2_p\}$.

Since each accepting path in an MSG defines an MSC, we can associate with an MSG $\mathfrak{G}$ a language $\mathcal{L}(\mathfrak{G})$ of MSCs. In general, it is undecidable to determine whether $\mathcal{L}(\mathfrak{G})$ is regular [7]. This is because processes move asynchronously along the MSC traced out by accepting paths and there is no bound, in general on this asynchrony. However, there is a sufficient structural condition to guarantee regularity [3, 10].

Given an MSC M, we construct its communication graph $CG(M)$ as follows: the vertices are the processes and we have a directed edge (p, q) if M contains a message from p to q. An MSC M is said to be *connected* if the non-isolated vertices in $CG(M)$ form a single strongly connected component. An MSG $\mathfrak{G}$ is said to be *locally synchronized* if for every loop π in $\mathfrak{G}$, the MSC $\Phi(\pi)$ is connected. Intuitively, this means that every message sent in a loop is implicitly acknowledged, because if p sends a message, there is a path in the communication graph back to p. This ensures that all channels are *universally bounded*—there is a uniform bound B such that across all linearisations, no channel ever has more than B pending messages. Thus, if $\mathfrak{G}$ is locally synchronized, $\mathcal{L}(\mathfrak{G})$ is a regular set of MSCs.

2.5 Time-constrained message sequence graphs

We generalise MSGs to the timed setting in a natural way. In a *time-constrained MSG (TCMSG)*, states are labelled by TCMSCs rather than basic MSCs. In addition, we also permit process-wise timing constraints along the edges of the graph. A constraint for process p along an edge $v \to v'$ specifies a constraint between the final p-event of $\Phi(v)$ and the initial p-event of $\Phi(v')$, provided p actively participates in both these nodes. If p does not participate in either of these nodes, the constraint is ignored. Formally, a TCMSG is a tuple $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ where $G = (V, \to, v_{in}, V_F)$ is a graph as before, $\Phi : V \to \mathcal{L}^{TC}$

labels each node with a TCMSC from a set $\mathcal{L}^{TC}$ and *EdgeC* associates a tuple of constraints with each edge—for convenience, we assume that any edge constraint not explicitly specified corresponds to the trivial constraint $(-\infty, \infty)$.

Each accepting path in a TCMSG defines a TCMSC. Given a path $v_0 v_1 \cdots v_n$, we concatenate the TCMSCs $\Phi(v_0), \Phi(v_1), \ldots, \Phi(v_n)$ and insert the additional constraints specified by *EdgeC*. We define $\mathcal{L}_{TC}(\mathfrak{G})$ to be the set of all TCMSCs over *Act* generated by accepting paths in G. We also let $\mathcal{L}_{time}(\mathfrak{G}) = \bigcup_{\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})} \mathcal{L}_{time}(\mathfrak{M})$ and $\mathcal{L}_{tw}(\mathfrak{G}) = \bigcup_{\mathfrak{M} \in \mathcal{L}_{TC}(\mathfrak{G})} \mathcal{L}_{tw}(\mathfrak{M})$. Figure 2 shows a TCMSG, a TCMSC that it generates and a realizing TMSC.

2.6 Timed automata

We can formulate many types of machine models for timed MSCs. One natural choice is a message-passing automaton (MPA) equipped with (local) clocks. In a timed MPA, we have one component for each process p, which is a finite state automaton over actions of the form $p!q(m)$ and $p?q(m)$. Each component also has local clocks that can be used to guard transitions. The global state space defines a timed automaton over *Act*.

A timed automaton over an alphabet Σ is a tuple $\mathcal{A} = (Q, \Delta, q_{in}, F, Z)$ where Q is a finite set of states, $q_{in} \in Q$ is the initial state, $F \subseteq Q$ are the final states and Z is a set of clocks that take values over $\mathbb{R}_{\geq 0}$. Each transition in Δ is of the form $q \xrightarrow{\varphi, a, X} q'$ where $q, q' \in Q$, $a \in \Sigma$, $X \subseteq Z$ and φ is a boolean combination of clock constraints of the form $x \ op \ c$ where $x \in Z$, $c \in \mathbb{Q}_{\geq 0}$ and $op \in \{\leq, <, >, \geq\}$. This transition is enabled if the current values of all clocks satisfy the guard φ. On taking this transition, the clocks in X are reset to 0. As is standard, time elapses between transitions, transitions occur instantaneously and such an automaton accepts timed words from TW_Σ. More details can be found in [2, 4].

For our purposes, we only need the following two results about timed automata.

- Given timed automata $\mathcal{A}_1$ and $\mathcal{A}_2$, we can construct a timed automaton $\mathcal{A}_{12}$ such that $L(\mathcal{A}_{12}) = L(\mathcal{A}_1) \cap L(\mathcal{A}_2)$.
- Checking whether the language of a timed automaton is empty is decidable.

2.7 The model checking problem

We are interested in timed automata over *Act* whose languages can be interpreted as timed MSCs. A timed word in TW_{Act} corresponds to a linearisation of a timed MSC provided the timed word is well-formed and complete. A word w over *Act* is *well-formed* if for each channel (p, q), in every prefix v of w, the sequence of messages received by q from p in v is a prefix of the messages sent from p to q in v. A well-formed word w is *complete* if $\#_{p!q}(w) = \#_{q?p}(w)$ for each matching pair of send-receive actions, where $\#_X(u)$ counts the number of occurrences in u of $X \subseteq Act$. Finally, a well-formed word w is B-bounded if, in every prefix v of w, $\#_{p!q}(v) - \#_{q?p}(v) \leq B$ for each channel (p, q). Correspondingly, a timed word is said to be well-formed (complete, B-bounded) if its projection onto *Act* is well-formed (complete, B-bounded). Well-formedness captures the intuition that any receive action has an earlier matching sending action. Completeness guarantees that all pending messages have been received. B-boundedness promises that no channel ever has more than B messages.

Given a timed automaton $\mathcal{A}$ over *Act* and a TCMSG specification $\mathfrak{G}$, the model checking problem is to check that every timed word accepted by $\mathcal{A}$ realises some TCMSC in $\mathcal{L}_{TC}(\mathfrak{G})$. Since $\mathcal{A}$ may accept timed words that are not well-formed or not complete, this implicitly includes checking that $\mathcal{A}$ accepts only well-formed and complete timed words in TW_{Act}.

From this, it is clear that the model checking problem corresponds to checking whether $L(\mathcal{A}) \subseteq \mathcal{L}_{tw}(\mathfrak{G})$. To make the problem tractable, we restrict our attention to locally synchronized TCMSGs, so that $\mathcal{L}_{tw}(\mathfrak{G})$ is a timed regular language. Unfortunately, checking inclusion is undecidable even for timed regular languages [2]. To get around this, we introduce a more restricted machine model for timed MSCs called MSC event clock automata, which are closed under complementation. It turns out that $\mathcal{L}_{tw}(\mathfrak{G})$ can be recognized by MSC event clock automata, yielding a solution to our model checking problem.

3 An extended event clock automaton – the MSC-ECA

We now define MSC event clock automata or MSC-ECA. These will be used to capture exactly the guards that occur in the TCMSGs that we have defined. We denote an MSC-ECA over Act by $\mathcal{C} = (Q, Act, \delta, q_0, F)$, with states Q, initial state $q_0 \in Q$ and final states $F \subseteq Q$. A transition in δ is of the form (q, φ, a, q') where $q, q' \in Q, a \in Act$ and φ is a conjunction of event clock guards, which are of two types: either $\mathrm{Y}_p^k \in I$ or $\mathrm{Msg}^{-1} \in I$, where I is an interval, as used in TCMSC timing constraints. We interpret these guards over timed words. Let $\sigma = (a_1, t_1) \cdots (a_n, t_n) \in \mathrm{TW}_{Act}$. Then at a position $1 \leq j \leq n$, we define

(D1) $\sigma, j \models \mathrm{Y}_p^k \in I$ if the time elapsed between the k^{th}-previous p-action a_i in σ and this action a_j is in the interval I.

(D2) $\sigma, j \models \mathrm{Msg}^{-1} \in I$ if a_j is a receive action and the time elapsed since the occurence of its matching send action a_i is in the interval I.

In both these definitions, note that action a_i is uniquely defined, i.e., there is at most one position i that matches a given position j with respect to a given event clock guard.

Now, we define runs of $\mathcal{C}$ over timed words. For a timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n)$, we say there is a run of $\mathcal{C}$ from q to q' on σ, denoted $q \xrightarrow{\sigma} q'$ *in* $\mathcal{C}$, if there exists a sequence of transitions $q = q_0 \xrightarrow{\varphi_1, a_1} \cdots \xrightarrow{\varphi_n, a_n} q_n$ such that for all j, $1 \leq j \leq n$, $\sigma, j \models \varphi_j$. The timed word σ is said to be accepted if it has a run from the initial to some final state in F. We denote by $\mathcal{L}_{tw}(\mathcal{C})$ the set of timed words accepted by the MSC-ECA $\mathcal{C}$.

3.1 Determinization and complementation of MSC-ECA

We now prove that MSC-ECA can be determinized and complemented, which is crucial for solving the model checking problem. We obtain this by constructing a deterministic and complete version of any given MSC-ECA. Intuitively, this works as for classical ECA's and the main reason is that there are no explicit clocks. Since the reset of an event clock only depends on the timed word being read and not on the path followed in the automaton, we can use the subset construction.

More precisely, let $\mathcal{C} = (Q, Act, \delta, q_0, F)$ be a finite MSC-ECA. The set of states of the universal automaton $\mathcal{C}^{univ}$ is 2^Q. For a set $X \subseteq Q$ and an action a, we let $T(X, a)$ denote the set of transitions in δ having action a and a source state in X. Then, for some $T' \subseteq T(X, a) = T$, we denote by $\mathrm{target}(T')$ the set of target states of transitions in T' and we define

$$\varphi(T', T) = \bigwedge_{t=(q,\varphi_t,a,q') \in T'} \varphi_t \wedge \bigwedge_{t=(q,\varphi_t,a,q') \in T \setminus T'} \neg \varphi_t \ .$$

Then, we denote the set of transitions of $\mathcal{C}^{univ}$ by Δ, where we say that $X \xrightarrow{\varphi, a} X' \in \Delta$ if there exists $T' \subseteq T = T(X, a)$ such that $\varphi = \varphi(T', T)$ and $X' = \mathrm{target}(T')$.

Note that, once we have fixed X, a and the set T', the transition is uniquely defined. Also for $X = \emptyset$, we have $T(X, a) = \emptyset$ and the only possible transition is $\emptyset \xrightarrow{\text{true},a} \emptyset$. The crucial property of $\mathcal{C}^{univ}$ is that it is deterministic and complete (and finite, if $\mathcal{C}$ is).

▶ **Lemma 1.** *Given any timed word $\sigma = (a_1, t_1) \cdots (a_n, t_n) \in \mathrm{TW}_{Act}$, there exists a unique run $X_0 \xrightarrow{\varphi_1,a_1} X_1 \xrightarrow{\varphi_2,a_2} \cdots X_{n-1} \xrightarrow{\varphi_n,a_n} X_n$ of $\mathcal{C}^{univ}$ on σ starting from $X_0 = \{q_0\}$. Moreover, $X_n = \{q \in Q \mid q_0 \xrightarrow{\sigma} q \text{ in } \mathcal{C}\}$.*

By suitably choosing the final states, $\mathcal{C}^{univ}$ will accept either the same language as $\mathcal{C}$ or its complement. Let $F_1 = \{X \in 2^Q \mid F \cap X \neq \emptyset\}$ and $F_2 = 2^Q \setminus F_1$. Define $\mathcal{C}_i^{univ} = (2^Q, Act, \Delta, \{q_0\}, F_i)$ for $i = \{1, 2\}$. From Lemma 1 we obtain:

▶ **Corollary 2.** *We have $\mathcal{L}_{tw}(\mathcal{C}_1^{univ}) = \mathcal{L}_{tw}(\mathcal{C})$ and $\mathcal{L}_{tw}(\mathcal{C}_2^{univ}) = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathcal{C})$.*

3.2 From MSC-ECA to TA

Not every MSC-ECA can be translated into an equivalent (classical) timed automaton. The problem comes from the event guards $\mathrm{Msg}^{-1} \in I$, which may require infinitely many clocks if channels are unbounded. Fortunately, thanks to the locally synchronized assumption on TCMSGs, we are only interested in bounded channels. Let $B > 0$. We show below how to construct a timed automaton $\mathcal{B}_{\mathcal{C}}^B$ from an MSC-ECA $\mathcal{C} = (Q, Act, \delta, q_0, F)$ such that $\mathcal{B}_{\mathcal{C}}^B$ and $\mathcal{C}$ are equivalent when restricted to B-bounded channels.

Let $K = \max\{k \mid \mathrm{Y}_p^k \in I \text{ occurs in some guard in } \delta\}$. A state of $\mathcal{B}_{\mathcal{C}}^B$ is either a dead state denoted $\perp$ or a tuple $\mathfrak{s} = (s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta})$ where $s \in Q$, $\overline{b} = (b_p)_{p \in Proc} \in \{0, 1\}^{Proc}$ ($b_p = 1$ if we have already seen at least K p-events), $\overline{n} = (n_p)_{p \in Proc} \in \{0, \ldots, K-1\}^{Proc}$ (n_p is the number of p-events already seen modulo K), $\overline{\alpha} = (\alpha_{p,q})_{p,q \in Proc} \in \{0, \ldots, B\}^{Proc^2}$ ($\alpha_{p,q}$ is the number of $q?p$ events modulo $B+1$), $\overline{\beta} = (\beta_{p,q})_{p,q \in Proc} \in \{0, \ldots B\}^{Proc^2}$ ($\beta_{p,q}$ is the number of $p!q$ events modulo $B+1$). The set of all states is denoted Q' and the initial state is $\mathfrak{s}_0 = (s_0, (0), (0), (0), (0))$. The set of clocks is $Y \cup Z$ where $Y = \{y_p^i \mid p \in Proc, 0 \leq i < K\}$ and $Z = \{z_{p,q}^i \mid p, q \in Proc, 0 \leq i \leq B\}$. We will reset clock y_p^i when executing the i^{th} p-event mod K. Also, $z_{p,q}^i$ will be reset when executing the i^{th} $p!q$ event mod $B+1$.

We say that channel (p, q) is *empty* if $\alpha_{p,q} = \beta_{p,q}$ and *full* if $\beta_{p,q} = \alpha_{p,q} + B \bmod (B+1)$. The set of transitions $\delta_{\mathcal{B}_{\mathcal{C}}^B}$ is defined as follows: Assume $s \xrightarrow{\varphi,a} s'$ in $\mathcal{C}$ with $a \in Act_p$. Then, we have three types of transitions in $\mathcal{B}_{\mathcal{C}}^B$. (Recall that $p!q$ and $p?q$ abbreviate all actions of the form $p!q(m)$ and $p?q(m)$, respectively.)

(Tr1) $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \xrightarrow{\text{true},a,\emptyset} \perp$ is in $\mathcal{B}_{\mathcal{C}}^B$ if either $a \in p!q$ and channel (p, q) is full (the bound was exceeded), or $a \in p?q$ and channel (p, q) is empty.

(Tr2) $(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \xrightarrow{\varphi',a,R} (s', \overline{b}', \overline{n}', \overline{\alpha}', \overline{\beta}')$ is in $\mathcal{B}_{\mathcal{C}}^B$ if we are not in the above case and the following conditions hold:

1. $b'_p = 1$ if $n_p = K - 1$ and $b'_p = b_p$ otherwise. Also, $b'_r = b_r$ for $r \neq p$.
2. $n'_p = (n_p + 1) \bmod K$ and $n'_r = n_r$ for $r \neq p$.
3. if $a \in p!q$, then $\beta'_{p,q} = (\beta_{p,q} + 1) \bmod (B+1)$ and $\beta'_{p',q'} = \beta_{p',q'}$ for $(p', q') \neq (p, q)$.

Also $\overline{\alpha}' = \overline{\alpha}$, $R = \{y_p^{n'_p}, z_{p,q}^{\beta'_{p,q}}\}$ and φ' is φ where $\mathrm{Y}_p^k \in I$ is replaced with

$$\begin{cases} \text{false} & \text{if } b_p = 0 \text{ and } k > n_p \\ y_p^{(K+n'_p-k) \bmod K} \in I & \text{otherwise} \end{cases}$$

4. if $a \in p?q$, then $\alpha'_{q,p} = \alpha_{q,p} + 1 \bmod (B+1)$ and $\alpha'_{q',p'} = \alpha_{q',p'}$ for $(q',p') \neq (q,p)$. Also $\overline{\beta}' = \overline{\beta}$, $R = \{y_p^{n'_p}\}$ and φ' is φ where $\mathrm{Y}_p^k \in I$ is replaced as above and $\mathrm{Msg}^{-1} \in I$ is replaced with $z_{q,p}^{\alpha'_{q,p}} \in I$.

(Tr3) $\bot \xrightarrow{\mathrm{true},a,\emptyset} \bot$ is in $\mathcal{B}_{\mathcal{C}}^B$ for all $a \in Act$.

In the following, we call a timed word w *weakly well-formed* (wwf) if for each channel (p,q), in every prefix v of w, we have $\#_{q?p}(w) \leq \#_{p!q}(w)$. This weak form does not require the send message sequence to be the same as the received one. Let $\mathrm{TW}_{Act}^{B,\mathrm{wf}}$ denote the set of timed words $\sigma \in \mathrm{TW}_{Act}$ which are both wwf and B-bounded. We can define different notions of acceptance (i.e., final states) on $\mathcal{B}_{\mathcal{C}}^B$ constructed from $\mathcal{C}$ to derive the results below.

▶ **Proposition 3.** *Let $\mathcal{C} = (Q, Act, \delta, q_0, F)$ and $\mathcal{B}_{\mathcal{C}}^B = (Q', Act, (Y \cup Z), \delta_{\mathcal{B}_{\mathcal{C}}^B})$ be as above.*

1. *With final states $F' = \{(s, \overline{b}, \overline{n}, \overline{\alpha}, \overline{\beta}) \mid s \in F\}$ the timed automaton $\mathcal{B}_{\mathcal{C}}^B$ accepts the language $\mathcal{L}_{tw}(\mathcal{C}) \cap \mathrm{TW}_{Act}^{B,\mathrm{wf}}$.*
2. *If $\mathcal{C}$ is complete (i.e., it has a run on every timed word over Act) then with final states $F'' = \{\bot\}$ the timed automaton $\mathcal{B}_{\mathcal{C}}^B$ accepts the complement of $\mathrm{TW}_{Act}^{B,\mathrm{wf}}$.*

Proof. (Sketch) Let $\sigma = (a_1,t_1)\cdots(a_m,t_m)$ be a wwf and B-bounded timed word. Consider a path $\pi = s_0 \xrightarrow{\varphi_1,a_1} s_1 \xrightarrow{\varphi_2,a_2} \cdots \xrightarrow{\varphi_m,a_m} s_m$ of $\mathcal{C}$. We can build inductively a path $\pi' = \mathfrak{s}_0 \xrightarrow{\varphi'_1,a_1,R_1} \mathfrak{s}_1 \xrightarrow{\varphi'_2,a_2,R_2} \cdots \xrightarrow{\varphi'_m,a_m,R_m} \mathfrak{s}_m$ of $\mathcal{B}_{\mathcal{C}}^B$ starting from its initial state $\mathfrak{s}_0$ and using *(Tr2)* only. Then, we can prove that if σ has a run through π in $\mathcal{C}$ (i.e., $\sigma, i \models \varphi_i$ for all $i \in \{1,\ldots,m\}$) then σ has a run through π' in $\mathcal{B}_{\mathcal{C}}^B$. Hence we obtain one inclusion of (1).

For the converse inclusion, we start with a path of $\mathcal{B}_{\mathcal{C}}^B$ starting from its initial state $\mathfrak{s}_0$ and which does not reach $\bot$: $\pi' = \mathfrak{s}_0 \xrightarrow{\varphi'_1,a_1,R_1} \mathfrak{s}_1 \xrightarrow{\varphi'_2,a_2,R_2} \cdots \xrightarrow{\varphi'_m,a_m,R_m} \mathfrak{s}_m$. Since we did not reach $\bot$, the timed word $\sigma = (a_1,t_1)\cdots(a_m,t_m)$ must be wwf and B-bounded. Moreover, transitions in π' comes from *(Tr2)* only and we can recover a corresponding path $\pi = s_0 \xrightarrow{\varphi_1,a_1} s_1 \xrightarrow{\varphi_2,a_2} \cdots \xrightarrow{\varphi_m,a_m} s_m$ in $\mathcal{C}$. Again, we can prove that if σ has a run through π' in $\mathcal{B}_{\mathcal{C}}^B$ then σ has a run through π in $\mathcal{C}$.

Statement (2) can be proved easily. ◀

4 From a locally synchronized TCMSG to a finite MSC-ECA

The main result is that locally synchronized TCMSGs define timed regular languages.

▶ **Theorem 4.** *If $\mathfrak{G} = (G, \mathcal{L}^{TC}, \Phi, EdgeC)$ is a locally synchronized TCMSG, then there exists a finite MSC-ECA $\mathcal{C}$, such that $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G})$.*

In the untimed case, the corresponding result has been stated and proved in different ways [3, 5, 6, 10]. We describe a different proof that is more suitable for the timed version.

We want to simulate the global run of a TCMSG by keeping a finite amount of information in the states of the MSC-ECA. Intuitively, we keep the sequence of nodes along the TCMSG path that have been started but not completed (at least one executed event but not all). Since the TCMSG is locally synchronized, the number of such nodes is always bounded.

We replace segments of nodes in the TCMSG path that have not been started yet by a special gap symbol #. Nodes will be inserted at gaps whenever necessary, making sure that the sequential run of the MSC-ECA is compatible with the TCMSG path. In fact, the insertion must satisfy two conditions: (1) when we insert a node it must not conflict with the events that have already occurred in later nodes and (2) finally, after all insertions, we

do obtain a path in the MSG. The latter is done by checking that when we fill a gap the corresponding bordering nodes have an edge in the graph.

We also replace segments of fully executed nodes of a TCMSG path by the set of processes that have been active in these nodes, so that we ensure condition (1) above.

For a node u, let E^u be the set of events in the MSC labelling u. We define an *extended node* to be a pair (u, c) where $u \in V$ and $c \subseteq E^u$ is a cut of E^u that contains the events that have been executed in node u. For simplicity, we extend the set of vertices V with two dummy vertices $\triangleright$, $\triangleleft$ and add edges from $\triangleright$ to the initial vertex v_{in} and from every final vertex $v \in V_F$ to $\triangleleft$. We also set $E^{\triangleright} = \emptyset = E^{\triangleleft}$ so that for $u \in \{\triangleright, \triangleleft\}$, the only extended node is $(u, \emptyset)$. The set of all extended nodes is denoted *ExtNodes*. An extended node (u, c) is said to be *completed* if $c = E^u$. Note that $(\triangleright, \emptyset)$ and $(\triangleleft, \emptyset)$ are completed by default.

A state α of our new automaton $\mathcal{C}$ is a sequence of extended nodes, gaps and subsets of processes: $\alpha \in \Pi^*$ where $\Pi = \mathit{ExtNodes} \uplus \{\#\} \uplus 2^{\mathit{Proc}}$. The *initial* state is $\alpha_0 = (\triangleright, \emptyset)\#(\triangleleft, \emptyset)$. *Final* states are of the form $(\triangleright, \emptyset)P(\triangleleft, \emptyset)$ where $P \subseteq \mathit{Proc}$.

An *extended event* of $\alpha \in \Pi^*$ is a pair $(e, \alpha_1(u, c))$ where $e \in E^u$ and $\alpha_1(u, c) \preceq \alpha$—i.e., $\alpha_1(u, c)$ is a prefix of α. We say that the extended event $(e, \alpha_1(u, c))$ is *executed* in α if $e \in c$ and *enabled* in α if the following hold:

(E1) It has not been executed, i.e., $e \notin c$.
(E2) All events within the node which are below it (in the partial order) have been executed, i.e., for all $e' \in E^u$ with $e' <^u e$, we have $e' \in c$.
(E3) If e belongs to process p, then all p-events on any node occurring before this node in α have been executed, i.e., if $e \in E^u_p$ then for all $\alpha'_1(u', c') \preceq \alpha_1$, we have $E^{u'}_p \subseteq c'$.

We introduce notation to describe the set of processes that participate in nodes, paths or states. For a node $u \in V$, $\mathit{OProc}(u) = \{p \in \mathit{Proc} \mid E^u_p \neq \emptyset\}$ denotes the set of processes that participate (*occur*) in u. This is extended to V^* as a morphism. Also, with $\mathit{OProc}((u, c)) = \mathit{OProc}(u)$, $\mathit{OProc}(\#) = \emptyset$ and $\mathit{OProc}(P) = P$, it extends to Π^*. In addition, for $\beta \in \Pi^*$, $\mathit{EProc}(\beta)$ denoting the set of processes having *executed* events in β, is given by the morphism defined by $\mathit{EProc}((u, c)) = \{p \in \mathit{Proc} \mid E^u_p \cap c \neq \emptyset\}$, $\mathit{EProc}(\#) = \emptyset$ and $\mathit{EProc}(P) = P$.

Now, the transitions can be defined by saying that at any state α we can choose to execute an enabled (extended) event or add a new (extended) node in a gap of the state, in which case we must execute an enabled event on the new node.

We first define the *node insertion* operation as a macro $\alpha_1\#\alpha_2 \xrightarrow{u} \alpha'_1(u, \emptyset)\alpha'_2$ which is said to hold if

(I1) for every process that participates in u, there is no executed event in the segment α_2 on that process, i.e., $\mathit{OProc}(u) \cap \mathit{EProc}(\alpha_2) = \emptyset$.
(I2) $\alpha'_1 \in \{\alpha_1, \alpha_1\#\}$ and if $\alpha'_1 = \alpha_1$ then $\alpha_1 = \alpha''_1(v, c)$ and $v \to u$ in G.
(I3) $\alpha'_2 \in \{\alpha_2, \#\alpha_2\}$ and if $\alpha'_2 = \alpha_2$ then $\alpha_2 = (v, c)\alpha''_2$ and $u \to v$ in G.

Next, we explain how completed nodes are deleted from a state α. To check (I1) we need to preserve the set of executed processes, hence a completed node u will be replaced by $\mathit{OProc}(u)$. We also preserve (do not throw away) the nodes around a gap in α so that conditions (I2)–(I3) can still be checked. Finally, we preserve nodes that start an edge constraint which needs to be verified later (this is useful for guards defined in the transition relation below). Formally, we define the reduction as a rewrite operation $\alpha \xrightarrow{\mathit{redn}} \alpha'$. There are two rewrite rules:

(R1) $\alpha_1 P P' \alpha_2 \xrightarrow{\mathit{redn}} \alpha_1(P \cup P')\alpha_2$, i.e., two adjacent process sets can be merged.

(R2) $\alpha_1(v, E^v)\alpha_2 \xrightarrow{redn} \alpha_1 OProc(v)\alpha_2$ (a completed node is replaced by the set of processes participating in it) if the following hold:

(C2.1) $v \in V$, $\varepsilon \neq \alpha_1 \notin \Pi^*\#$, $\varepsilon \neq \alpha_2 \notin \#\Pi^*$ i.e., the node v is not next to a gap or at the beginning or the end.

(C2.2) (i) either the first symbol of α_2 is an extended node (v', c') and if both E^v_p and $E^{v'}_p$ are nonempty, then some event in $E^{v'}_p$ has occured (hence the edge constraint, if any, has already been checked),

(ii) or $\alpha_2 \in 2^{Proc}\Pi^*$ in which case there is no unchecked edge constraint.

▶ **Lemma 5.** *The rewrite system defined by the operation* $\xrightarrow{redn}$ *is confluent.*

Using the above lemma we conclude that, from any state α, after any maximal sequence of reductions, we reach the same state, which we denote by $Red(\alpha)$.

Now, we can define the transition relation: $\alpha \xrightarrow{\varphi,a} \alpha'$ is a transition in $\mathcal{C}$ if there exists $\beta = \beta_1(u, c)\beta_2$ and an extended event $(e, \beta_1(u, c))$ enabled in β such that

(i) either $\beta = \alpha$, i.e., the enabled event is already present in the current state,

(ii) or $\alpha = \alpha_1\#\alpha_2 \xrightarrow{u} \beta_1(u, \emptyset)\beta_2 = \beta$. Hence, $c = \emptyset$, $\beta_1 \in \{\alpha_1, \alpha_1\#\}$ and $\beta_2 \in \{\alpha_2, \#\alpha_2\}$

and all the following conditions hold:

(T1) $a = \lambda^u(e)$.

(T2) The guard φ checks all local and edge constraints—i.e.,

$$\varphi = \left(\bigwedge_{e' \in E^u, I \in \mathcal{I} \mid \tau^u(e', e) = I} \varphi(u, e', e, I) \right) \wedge \varphi^{edge} \text{ where,} \tag{1}$$

$$\varphi(u, e', e, I) = \begin{cases} \mathrm{Msg}^{-1} \in I & \text{if } \exists p, q, p \neq q \text{ s.t. } e' <^u_{qp} e \\ \mathrm{Y}^k_p \in I & \text{if } e, e' \in E^u_p \text{ and } |\{e'' \in E^u_p \mid e' \leq^u_{pp} e'' <^u_{pp} e\}| = k \end{cases} \tag{2}$$

$$\text{and } \varphi^{edge} = \begin{cases} \mathrm{Y}^1_p \in I & \text{if } \beta_1 = \beta'_1(u', c'') \text{ and for some } p \in Proc, \text{ we have} \\ & EdgeC((u', u), p) = I \text{ and } e = min(E^u_p) \\ \text{true} & \text{otherwise} \end{cases} \tag{3}$$

(T3) $\alpha' = Red(\beta_1(u, c')\beta_2)$ where $c' = c \uplus \{e\}$.

Observe that, once the state and the enabled event which is to be executed are fixed, the transition that is taken and indeed the state reached after the transition are uniquely determined. We can also observe that every reachable state α of $\mathcal{C}$ is *valid*. By this we mean that it satisfies the following properties:

(V1) Every # symbol in α is surrounded by nodes from *ExtNodes*. Also α starts with $(\triangleright, \emptyset)$ and ends with $(\triangleleft, \emptyset)$.

(V2) For any two consecutive extended nodes in α, there exists an edge between the nodes in G, i.e., for all $\alpha_1(u, c)(u', c') \preceq \alpha$, we have $u \to u'$ in G.

(V3) Executed events in α are *downward closed*:

a. For all $\alpha_1(u, c) \preceq \alpha$, if $e \in c$ and $e' \leq^u e$ then $e' \in c$.

b. For all $\alpha_1(u, c)\alpha_2(u', c') \preceq \alpha$, if $e \in E^u_p$ and $e' \in c' \cap E^{u'}_p$ for some p, then $e \in c$.

In order to get finiteness of the automaton $\mathcal{C}$, we need to restrict to states that are both reachable and *completable*. Formally, we call a state α *completable* if whenever $\alpha = \alpha_1(u,c)\#(v,c')\alpha_2$, there is $\beta \in V^+$ such that $u\beta v$ is a path in G and $OProc(\beta) \cap EProc((v,c')\alpha_2) = \emptyset$. Note that, in order to be co-reachable in $\mathcal{C}$, a state must be completable.

▶ **Lemma 6.** *If $\mathfrak{G}$ is locally synchronized, the set of states of $\mathcal{C}$ which are both valid and completable is finite.*

Proof. (Sketch) It is enough to show that the length of each valid, completable state of $\alpha \in \Pi^*$ is bounded. By definition, every extended node in α has at least one executed event. Using the locally synchronized assumption, one can prove the following properties about a loop in a state.

▶ Claim 7. Let $\alpha(u,c)\beta(u,c')\gamma$ be a valid completable state of $\mathcal{C}^{fin}_{\mathfrak{G}}$. If $(u,c)\beta$ is not completely executed or if $\#$ occurs in β, then $EProc((u,c')\gamma) \subsetneq EProc((u,c)\beta(u,c')\gamma)$.

Now, consider a loop $\alpha(u,c)\beta(u,c')\gamma$ in a valid completable state. If β has no $\#$ and $(u,c)\beta$ is completely executed, then $\alpha = \alpha'\#$. Indeed, otherwise the completed node (u,c) would have been deleted. Along with the previous claim this implies that we can bound the number of occurences of a node u in a path by $2|Proc|$. From which we can conclude that we have a bound of $2|Proc||V|$ on the number of extended nodes in a path. But we know that each $\#$ or $P \subseteq Proc$ must have an extended node next to it on the left. So we can conclude that the length of the path is $\mathcal{O}(|Proc||V|)$. Thus $\mathcal{C}$ is finite. ◀

The main result is stated in the following proposition.

▶ **Proposition 8.** $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G})$.

The proof which is long and technical is omitted for lack of space. It can be found in [1] where it is split in three main steps. First we construct an MSC-ECA with infinitely many states: we guess the full path of the TCMSG initially and we keep it in all states along the run to avoid the complication of node insertions and node deletions. Next, we introduce the automaton with gaps, dealing with node insertions but not yet with node deletions. This automaton is still infinite. Finally we introduce node deletions to obtain the automaton $\mathcal{C}$ constructed above. At each step we prove the equality of the timed languages, either directly, or using bisimulation at the abstract level of paths.

5 Solving the model checking problem

Now, we are in a position to solve the model checking problem.

▶ **Theorem 9.** *For a locally synchronized TCMSG $\mathfrak{G}$ and a timed automaton $\mathcal{A}$, the model checking problem $\mathcal{L}_{tw}(\mathcal{A}) \subseteq \mathcal{L}_{tw}(\mathfrak{G})$ is decidable, i.e., it is decidable to check if for all timed words σ generated by $\mathcal{A}$ there exists some $\mathfrak{M}$ specified by $\mathfrak{G}$ such that σ is a linearisation of a TMSC T which realises $\mathfrak{M}$.*

Proof. We have to prove that $\mathcal{L}_{tw}(\mathcal{A}) \cap (\mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G})) = \emptyset$. By Theorem 4 we can construct an MSC-ECA $\mathcal{C}$ such that $\mathcal{L}_{tw}(\mathcal{C}) = \mathcal{L}_{tw}(\mathfrak{G})$. Using the complementation construction of Section 3.1 we can build a deterministic and complete MSC-ECA $\mathcal{C}' = \mathcal{C}_2^{univ}$ such that by Corollary 2 we have $\mathcal{L}_{tw}(\mathcal{C}') = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathcal{C}) = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G})$.

Since $\mathfrak{G}$ is locally synchronized, there is a bound $B > 0$ such that each timed word $\sigma \in \mathcal{L}_{tw}(\mathfrak{G})$ is wwf and B-bounded: $\mathcal{L}_{tw}(\mathfrak{G}) \subseteq \mathrm{TW}^{B,\mathrm{wf}}_{Act}$. Consider the timed automaton $\mathcal{B}^B_{\mathcal{C}'}$

associated with $\mathcal{C}'$ and the bound B by the construction of Section 3.2. For final states of $\mathcal{B}^B_{\mathcal{C}'}$ we choose $F' \cup F''$ as defined in Proposition 3. We get $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}'}) = (\mathrm{TW}_{Act} \setminus \mathrm{TW}^{B,\mathrm{wf}}_{Act}) \cup (\mathcal{L}_{tw}(\mathcal{C}') \cap \mathrm{TW}^{B,\mathrm{wf}}_{Act}) = (\mathrm{TW}_{Act} \setminus \mathrm{TW}^{B,\mathrm{wf}}_{Act}) \cup (\mathrm{TW}^{B,\mathrm{wf}}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G}))$. Using $\mathcal{L}_{tw}(\mathfrak{G}) \subseteq \mathrm{TW}^{B,\mathrm{wf}}_{Act}$ we deduce $\mathcal{L}_{tw}(\mathcal{B}^B_{\mathcal{C}'}) = \mathrm{TW}_{Act} \setminus \mathcal{L}_{tw}(\mathfrak{G})$.

Hence, the model checking problem is reduced to checking emptiness of the intersection of two timed automata, $\mathcal{A}$ and $\mathcal{B}^B_{\mathcal{C}'}$, which is indeed decidable. ◀

References

1. S. Akshay, P. Gastin, M. Mukund and K. Narayan Kumar: Model checking time-constrained scenario-based specifications. Technical Report LSV-10-16, ENS Cachan, 2010. Available at `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/rapports`.
2. R. Alur and D. Dill: A Theory of Timed Automata. *Theor. Comput. Sci.*, **126** (1994) 183–225.
3. R. Alur and M. Yannakakis: Model checking of message sequence charts. *Proc. CONCUR'99*, Springer LNCS **1664** (1999) 114–129
4. J. Bengtsson and Wang Yi: Timed Automata: Semantics, Algorithms and Tools, *Lectures on Concurrency and Petri Nets 2003*, Springer LNCS **3098** (2003) 87–124.
5. J. Chakraborty, D. D'Souza, and K. Narayan Kumar. Analysing message sequence graph specifications. Technical Report IISc-CSA-TR-2009-1, IISc Bangalore, 2009.
6. M. Clerbout and M. Latteux. Semi-commutations. *Inf. Comp.*, **73(1)** (1987) 59–74.
7. J.G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P.S. Thiagarajan: A Theory of Regular MSC Languages. *Inf. Comp.*, **202(1)** (2005) 1–38.
8. ITU-T Recommendation Z.120: *Message Sequence Chart (MSC)*. ITU, Geneva (1999).
9. S. Mauw and M.A. Reniers: High-level message sequence charts. *Proc. SDL'97*, Elsevier (1997) 291–306.
10. A. Muscholl and D. Peled: Message sequence graphs and decision problems on Mazurkiewicz traces. *Proc. MFCS'99*, Springer LNCS **1672** (1999) 81–91.

Global Model Checking of Ordered Multi-Pushdown Systems

Mohamed Faouzi Atig

Uppsala University, Sweden
mohamed_faouzi.atig@it.uu.se

Abstract

In this paper, we address the verification problem of ordered multi-pushdown systems: A multi-stack extension of pushdown systems that comes with a constraint on stack operations such that a pop can only be performed on the first non-empty stack. First, we show that for an ordered multi-pushdown system the set of all predecessors of a regular set of configurations is an effectively constructible regular set. Then, we exploit this result to solve the global model checking which consists in computing the set of all configurations of an ordered multi-pushdown system that satisfy a given w-regular property (expressible in linear-time temporal logics or the linear-time μ-calculus). As an immediate consequence of this result, we obtain an 2ETIME upper bound for the model checking problem of w-regular properties for ordered multi-pushdown systems (matching its lower-bound).

Keywords and phrases Concurrent Programs, Pushdown Systems, Global Model-Checking.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.216

1 Introduction

Automated verification of multi-threaded programs is an important and a highly challenging problem. In fact, even when such programs manipulate data ranging over finite domains, their control structure can be complex due to the handling of (recursive) procedure calls in the presence of concurrency and synchronization between threads.

In the last few years, a lot of effort has been devoted to the verification problem for models of concurrent programs (see, e.g., [7, 24, 15, 2, 25, 3, 13, 16]) where each thread corresponds to a sequential program with (recursive) procedure calls. In fact, it is well admitted that pushdown systems are adequate models for such kind of threads [10, 21], and therefore, it is natural to model recursive concurrent programs as multi-stack systems.

In general, multi-stack systems are Turing powerful and hence come along with undecidability of basic decision problems [20]. A lot of efforts have been nevertheless devoted recently to the development of precise analysis algorithms of specific formal models of some classes of programs [17, 11, 8, 22, 14].

Context-bounding has been proposed in [19] as a suitable technique for the analysis of multi-stack systems. The idea is to consider only runs of the system that can be divided into a given number of contexts, where in each context pop and push operations are exclusive to one stack. The state space which may be explored is still unbounded in presence of recursive procedure calls, but the context-bounded reachability problem is NP-complete even in this case. In fact, context-bounding provides a very useful tradeoff between computational complexity and verification coverage.

In [24], La Torre et al. propose a more general definition of the notion of a context. For that, they define the class of *bounded-phase visibly multi-stack pushdown systems* (BVMPS) where only those runs are taken into consideration that can be split into a given number of phases, where each phase admits pop operations of one particular stack only. In the above

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 216–227

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

case, the emptiness problem is decidable in double exponential time by reducing it to the emptiness problem for tree automata.

Another way to regain decidability is to impose some order on stack operations. In [9], Breveglieri et al. define *ordered multi-pushdown systems* (OMPS), which impose a linear ordering on stacks. Stack operations are constrained in such a way that a pop operation is reserved to the first non-empty stack. In [1], we show that the emptiness problem for OMPS is in 2ETIME-complete. (Recall that 2ETIME is the class of all decision problems solvable by a deterministic Turing machine in time $2^{2^{dn}}$ for some constant d.) The proof of this result lies in a complex encoding of OMPS into some class of grammars for which the emptiness problem is decidable. Moreover, we prove that the class of ordered multi-pushdown systems with $2k$ stacks are strictly more expressive than bounded-phase visibly multi-stack pushdown systems with k phases.

In this paper, we consider the problem of verifying ordered multi-pushdown systems with respect to a given w-regular property (expressible in the linear-time temporal logics [18] or the linear-time μ-calculus [26]). In particular, we are interested in solving the global model checking for ordered multi-pushdown systems which consists in computing the set of all configurations that satisfy a given w-regular property. The basic ingredient for achieving this goal is to define a procedure for computing the set of backward reachable configurations from a given set of configurations. Therefore, our first task is to find a finite symbolic representation of the possibly infinite state-space of an ordered multi-pushdown system. For that, we consider the class of recognizable sets of configurations defined using finite state automata [19, 2, 23].

Then, we show that for an ordered multi-pushdown system $\mathcal{M}$ the set of all predecessors $Pre^*(C)$ of a recognizable set of configurations C is an effectively constructible recognizable set. The proof of this result is done by induction on the number of stacks of $\mathcal{M}$. Technically, we use a result given in [4] establishing that the set of configurations C_n, where the first $(n-1)$ stacks are empty, from which $\mathcal{M}$ can reach a configuration in C is recognizable and effectively constructible. Then, to compute the intermediary configurations in $Pre^*(C)$ when the first $(n-1)$ stacks are not empty, we construct an ordered multi-pushdown system $\mathcal{M}'$ with $(n-1)$ stacks that: (1) performs the same operations on its stacks as the ones performed by $\mathcal{M}$ on its first $(n-1)$ stacks, and (2) simulates a push operation of $\mathcal{M}$ over its n-th stack by a transition of the finite-state automaton accepting the recognizable set of configurations C_n. Now, we can apply the induction hypothesis to $\mathcal{M}'$ and construct a finite-state automaton accepting the set of all predecessors $Pre^*(C)$.

As an application of this result, we show that the set of configurations of an ordered multi-pushdown system satisfying a given w-regular property is recognizable and effectively constructible. Our approach also allows to obtain an 2ETIME upper bound for the model checking problem of w-regular properties for ordered multi-pushdown systems (matching its lower-bound [1]).

Related works: In [23], A. Seth shows that the set of predecessors of a recognizable set of configurations of a bounded-phase visibly multi-stack pushdown system is recognizable and effectively constructible. In fact, our results generalize the obtained result in [23] since any bounded-phase visibly multi-stack pushdown system with k phases can be simulated by an ordered multi-pushdown system with $2k$ stacks [1].

To the best of our knowledge, this is the first work that addresses the global model checking for ordered multi-pushdown systems.

2 Preliminaries

In this section, we introduce some basic definitions and notations that will be used in the rest of the paper.

Integers: Let $\mathbb{N}$ be the set of natural numbers. For every $i, j \in \mathbb{N}$ such that $i \leq j$, we use $[i, j]$ (resp. $[i, j[$) to denote the set $\{k \in \mathbb{N} \mid i \leq k \leq j\}$ (resp. $\{k \in \mathbb{N} \mid i \leq k < j\}$).

Words and languages: Let Σ be a finite alphabet. We denote by Σ^* (resp. Σ^+) the set of all words (resp. non empty words) over Σ, and by ϵ the empty word. A language is a (possibly infinite) set of words. We use Σ_ϵ to denote the set $\Sigma \cup \{\epsilon\}$.

Let u be a word over Σ. The length of u is denoted by $|u|$. For every $j \in [1, |u|]$, we use $u(j)$ to denote the j^{th} letter of u. We denote by u^R the mirror of u.

Transition systems: A transition system (TS for short) is a triplet $\mathcal{T} = (C, \Sigma, \rightarrow)$ where: (1) C is a (possibly infinite) set of configurations, (2) Σ is a finite set of labels (or actions) such that $C \cap \Sigma = \emptyset$, and (3) $\rightarrow \subseteq C \times \Sigma_\epsilon \times C$ is a transition relation. We write $c \xrightarrow{a}_{\mathcal{T}} c'$ whenever c and c' are two configurations and a is an action such that $(c, a, c') \in \rightarrow$.

Given two configurations $c, c' \in C$, a finite run ρ of $\mathcal{T}$ from c to c' is a finite sequence $c_0 a_1 c_1 \cdots a_n c_n$, for some $n \geq 1$, such that: (1) $c_0 = c$ and $c_n = c'$, and (2) $c_i \xrightarrow{a_{i+1}}_{\mathcal{T}} c_{i+1}$ for all $i \in [0, n[$. In this case, we say that ρ has length n and is labelled by the word $a_1 a_2 \cdots a_n$.

Let $c, c' \in C$ and $u \in \Sigma^*$. We write $c \overset{u}{\underset{n}{\Longrightarrow}}_{\mathcal{T}} c'$ if one of the following two cases holds: (1) $n = 0$, $c = c'$, and $u = \epsilon$, and (2) there is a run ρ of length n from c to c' labelled by u. We also write $c \overset{u}{\Longrightarrow}^*_{\mathcal{T}} c'$ (resp. $c \overset{u}{\Longrightarrow}^+_{\mathcal{T}} c'$) to denote that $c \overset{u}{\underset{n}{\Longrightarrow}}_{\mathcal{T}} c'$ for some $n \geq 0$ (resp. $n > 0$).

For every $C_1, C_2 \subseteq C$, let $Traces_{\mathcal{T}}(C_1, C_2) = \{u \in \Sigma^* \mid \exists (c_1, c_2) \in C_1 \times C_2 \,,\, c_1 \overset{u}{\Longrightarrow}^*_{\mathcal{T}} c_2\}$ be the set of sequences of actions generated by the runs of $\mathcal{T}$ from a configuration in C_1 to a configuration in C_2.

For every $C' \subseteq C$, let $Pre_{\mathcal{T}}(C') = \{c \in C \mid \exists (c', a) \in C' \times \Sigma_\epsilon \,,\, c \xrightarrow{a}_{\mathcal{T}} c'\}$ be the set of immediate predecessors of C'. Let $Pre^*_{\mathcal{T}}$ be the reflexive-transitive closure of $Pre_{\mathcal{T}}$, and let $Pre^+_{\mathcal{T}} = Pre_{\mathcal{T}} \circ Pre^*_{\mathcal{T}}$.

Finite state automata: A finite state automaton (FSA) is a tuple $\mathcal{A} = (Q, \Sigma, \Delta, I, F)$ where: (1) Q is the finite non-empty set of states, (2) Σ is the finite input alphabet, (3) $\Delta \subseteq (Q \times \Sigma_\epsilon \times Q)$ is the transition relation, (4) $I \subseteq Q$ is the set of initial states, and (5) $F \subseteq Q$ is the set of final states. We represent a transition (q, a, q') in Δ by $q \xrightarrow{a}_{\mathcal{A}} q'$. Moreover, if I' and F' are two subsets of Q, then we use $\mathcal{A}(I', F')$ to denote the finite state automaton defined by the tuple $(Q, \Sigma, \Delta, I', F')$.

The size of $\mathcal{A}$ is defined by $|\mathcal{A}| = (|Q| + |\Sigma|)$. We use $\mathcal{T}(\mathcal{A}) = (Q, \Sigma, \Delta)$ to denote the transition system associated with $\mathcal{A}$. The language accepted (or recognized) by $\mathcal{A}$ is given by $L(\mathcal{A}) = Traces_{\mathcal{T}(\mathcal{A})}(I, F)$.

3 Ordered Multi-Pushdown Systems

In this section, we first recall the definition of *multi-pushdown systems*. Then *ordered multi-pushdown systems* [9, 1] appear as a special case of multi-pushdown systems.

3.1 Multi-pushdown systems

Multi-pushdown systems have one read-only left to right input tape and $n \geq 1$ read-write memory tapes (stacks) with a last-in-first-out rewriting policy. A transition is of the form $t = \langle q, \gamma_1, \ldots, \gamma_n \rangle \rightarrow \langle q', \alpha_1, \ldots, \alpha_n \rangle$. Being in a configuration $(p, w_1, \ldots, w_n)$, which is composed of a state p and a stack content w_i for each stack i, t can be applied if both $q = p$ and the i-th stack is of the form $\gamma_i w'_i$ for some w'_i. Taking the transition, the system moves to the successor configuration $(q', \alpha_1 w'_1, \ldots, \alpha_n w'_n)$.

▶ **Definition 1.** A *multi-pushdown system* (MPS) is a tuple $\mathcal{M} = (n, Q, \Gamma, \Delta)$ where $n \geq 1$ is the number of stacks, Q is the finite set of *states*, Γ is the *stack alphabet* containing the special stack symbol $\perp$, and $\Delta \subseteq \big(Q \times (\Gamma_\epsilon)^n\big) \times \big(Q \times (\Gamma^*)^n\big)$ is the *transition relation* such that, for all $((q, \gamma_1, \ldots, \gamma_n), (q', \alpha_1, \ldots, \alpha_n)) \in \Delta$ and $i \in [1, n]$, we have:

- $|\alpha_i| \leq 2$.
- If $\gamma_i \neq \perp$, then $\alpha_i \in (\Gamma \setminus \{\perp\})^*$.
- If $\gamma_i = \perp$, then $\alpha_i = \alpha'_i \perp$ for some $\alpha'_i \in \Gamma_\epsilon$.

In the rest of this paper, we use $\langle q, \gamma_1, \ldots, \gamma_n \rangle \rightarrow_{\mathcal{M}} \langle q', \alpha_1, \ldots, \alpha_n \rangle$ to denote that the transition $((q, \gamma_1, \ldots, \gamma_n), (q', \alpha_1, \ldots, \alpha_n))$ is in Δ. The size of $\mathcal{M}$, denoted by $|\mathcal{M}|$, is defined by $(n + |Q| + |\Sigma| + |\Gamma|)$.

A stack content of $\mathcal{M}$ is a sequence from $\mathit{Stack}(\mathcal{M}) = (\Gamma \setminus \{\perp\})^* \{\perp\}$. A configuration of $\mathcal{M}$ is a $(n+1)$-tuple $(q, w_1, \ldots, w_n)$ with $q \in Q$ and $w_1, \ldots, w_n \in \mathit{Stack}(\mathcal{M})$. The set of all configurations of $\mathcal{M}$ is denoted by $\mathit{Conf}(\mathcal{M})$.

The behavior of the MPS $\mathcal{M}$ is described by its corresponding TS $\mathcal{T}(\mathcal{M})$ defined by the tuple $(\mathit{Conf}(\mathcal{M}), \Sigma, \rightarrow)$ where $\Sigma = \Delta$ and $\rightarrow$ is the smallest transition relation such that if $t = \langle q, \gamma_1, \ldots, \gamma_n \rangle \rightarrow_{\mathcal{M}} \langle q', \alpha_1, \ldots, \alpha_n \rangle$ then $(q, \gamma_1 w_1, \ldots, \gamma_n w_n) \xrightarrow{t}_{\mathcal{T}(\mathcal{M})} (q', \alpha_1 w_1, \ldots, \alpha_n w_n)$ for all $w_1, \ldots, w_n \in \Gamma^*$ such that $\gamma_1 w_1, \ldots, \gamma_n w_n \in \mathit{Stack}(\mathcal{M})$. Observe that the symbol $\perp$ marks the bottom of a stack. According to the transition relation, $\perp$ can never be popped.

3.2 Symbolic representation of MPS configurations

We show in this section how we can symbolically represent infinite sets of MPS configurations using special kind of finite automata which were introduced in [23]. Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be a MPS. A $\mathcal{M}$-automaton for accepting configurations of $\mathcal{M}$ is a finite state automaton $\mathcal{A} = (Q_{\mathcal{M}}, \Gamma, \Delta_{\mathcal{M}}, I_{\mathcal{M}}, F_{\mathcal{M}})$ such that $I_{\mathcal{M}} = Q$. We say that a configuration $(q, w_1, \ldots, w_n)$ of $\mathcal{M}$ is accepted (or recognized) by $\mathcal{A}$ if and only if the word $w = w_1 w_2 \cdots w_n$ is in $L(\mathcal{A}(\{q\}, F_{\mathcal{M}}))$. (Notice that for every word $w \in L(\mathcal{A}(\{q\}, F_{\mathcal{M}}))$ there are unique words $w_1, \ldots, w_n \in \mathit{Stack}(\mathcal{M})$ such that $w = w_1 \cdots w_n$.) The set of all configurations recognized by $\mathcal{A}$ is denoted by $L_{\mathcal{M}}(\mathcal{A})$. A set of configurations of $\mathcal{M}$ is said to be recognizable if and only if it is accepted by some $\mathcal{M}$-automaton.

Finally, it is easy to see that the class of $\mathcal{M}$-automaton is closed under boolean operations and that the emptiness and membership problems are decidable in polynomial time.

3.3 Ordered multi-pushdown systems

An ordered multi-pushdown system is a multi-pushdown system in which one can pop only from the first non-empty stack (i.e., all preceding stacks are equal to $\perp$).

▶ **Definition 2.** An ordered multi-pushdown system (OMPS for short) is a multi-pushdown system (n, Q, Γ, Δ) such that Δ contains only the following types of transitions:

- $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \rightarrow_{\mathcal{M}} \langle q', \gamma''\gamma', \epsilon, \ldots, \epsilon\rangle$ for some $q, q' \in Q$ and $\gamma, \gamma', \gamma'' \in (\Gamma \setminus \{\bot\})$.
- $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \rightarrow_{\mathcal{M}} \langle q', \epsilon, \ldots, \epsilon, \gamma', \epsilon, \ldots, \epsilon\rangle$ for some $q, q' \in Q$ and $\gamma, \gamma' \in (\Gamma \setminus \{\bot\})$ (γ' is pushed on one of stacks 2 to n).
- $\langle q, \bot, \ldots, \bot, \gamma, \epsilon, \ldots, \epsilon\rangle \rightarrow_{\mathcal{M}} \langle q', \gamma'\bot, \bot, \ldots, \bot, \epsilon, \ldots, \epsilon\rangle$ for some $q, q' \in Q$ and $\gamma, \gamma' \in (\Gamma \setminus \{\bot\})$ (γ is popped from one of the stacks 2 to n).
- $\langle q, \gamma, \epsilon, \ldots, \epsilon\rangle \rightarrow_{\mathcal{M}} \langle q', \epsilon, \ldots, \epsilon\rangle$ for some $q, q' \in Q$ and $\gamma \in (\Gamma \setminus \{\bot\})$.

For $n \geq 1$, we call a MPS (resp. OMPS) a n-MPS (resp. n-OMPS) if its number of stacks is equal to n.

4 Computing the set of predecessors for an OMPS

In this section, we show that the set of predecessors of a recognizable set C of configurations of an OMPS is recognizable and effectively constructible (see Corollary 8). To simplify the presentation, we can assune without loss of generality that the set C contains only one configuration of the form $(q_f, \bot, \ldots, \bot)$ where all the stacks are empty. This result is established by Lemma 3.

▶ **Lemma 3.** *Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS and $\mathcal{A}$ be a $\mathcal{M}$-automaton. Then, it is possible to construct, in time and space polynomial in $(|\mathcal{M}| + |\mathcal{A}|)$, an OMPS $\mathcal{M}' = (n, Q' \cup \{q_f\}, \Gamma, \Delta')$ where $Q \subseteq Q'$, $q_f \notin Q'$, and $|\mathcal{M}'| = O(|\mathcal{M}| + |\mathcal{A}|)$ such that for every $c \in Conf(\mathcal{M})$, $c \in Pre^*_{\mathcal{T}(\mathcal{M})}(L_{\mathcal{M}}(\mathcal{A}))$ if and only if $c \in Pre^*_{\mathcal{T}(\mathcal{M}')}(\{(q_f, \bot, \ldots, \bot)\})$.*

Proof. The proof is similar to the case of standard pushdown systems. Technically, this can be done by adding to the OMPS $\mathcal{M}$ pop rules that check, in nondeterministic way, if the current configurations belongs to $L_{\mathcal{M}}(\mathcal{A})$ by simulating the finite state automaton $\mathcal{A}$. ◀

In the following, we recall a result given in [4] establishing that the set of configurations C' with empty first $(n-1)$ stacks (i.e., $C' \subseteq Q \times (\{\bot\})^{n-1} \times Stack(\mathcal{M})$) from which the OMPS $\mathcal{M}$ can reach a configuration of the form $(q, \bot, \ldots, \bot)$ where all the stacks are empty is recognizable and effectively constructible.

▶ **Lemma 4.** *Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS and $q \in Q$ be a state. Then, it is possible to construct, in time $O(|\mathcal{M}|^{2^{dn}})$ with d is a constant, a $\mathcal{M}$-automaton $\mathcal{A}$ such that $|\mathcal{A}| = O(|\mathcal{M}|)$ and $c \in L_{\mathcal{M}}(\mathcal{A})$ if and only if $c \in Pre^*_{\mathcal{T}(\mathcal{M})}(\{(q, \bot, \ldots, \bot)\})$ and $c = (q', \bot, \ldots, \bot, w)$ for some $q' \in Q$ and $w \in Stack(\mathcal{M})$.*

Proof. To prove Lemma 4 in [4], we have defined the class of *effective generalized pushdown systems* (EGPS) where operations on stacks are (1) pop the top symbol of the stack, and (2) push a word in some (effectively) given set of words L over the stack alphabet, assuming that L is in some class of languages for which checking whether L intersects regular languages is decidable. We have shown in [4] that the automata-based saturation procedure for computing the set of predecessors in standard pushdown systems [5] can be extended to prove that for EGPS too the set of all predecessors of a recognizable set of configurations is an effectively constructible recognizable set.

Then, we have shown that, given an OMPS $\mathcal{M}$ with n stacks, it is possible to construct an EGPS $\mathcal{P}$, whose pushed languages are defined by OMPSs with $(n-1)$ stacks, such that the following invariant is preserved: The state and the stack's content of $\mathcal{P}$ are the same as the state and the content of the n-th stack of $\mathcal{M}$ when its first $(n-1)$ stacks are empty. Thus, the saturation procedure for EGPS can be used to show that Lemma 4 holds. ◀

Next, we state our main theorem which is a generalization of the result obtained in [23].

▶ **Theorem 5.** *Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS and $q \in Q$ be a state. Then, it is possible to construct, in time $O(|\mathcal{M}|^{2^{dn}})$ where d is a constant, a $\mathcal{M}$-automaton $\mathcal{A}$ such that $|\mathcal{A}| = O(|\mathcal{M}|^{2^{dn}})$ and $L_{\mathcal{M}}(\mathcal{A}) = Pre^*_{\mathcal{T}(\mathcal{M})}(\{(q, \bot, \ldots, \bot)\})$.*

Proof. We proceed by induction on the number of stacks of the OMPS $\mathcal{M}$.

Basis. $n = 1$. Then, $\mathcal{M}$ is a pushdown system. From [5], we know that the emptiness problem for $\mathcal{M}$ can be solved in time polynomial in $|\mathcal{M}|$.

Step. $n > 1$. Then, we can use Lemma 4 to construct, in time $O(|\mathcal{M}|^{2^{d'n}})$ with d' is a constant (we assume w.l.o.g that $d' < d$), a $\mathcal{M}$-automaton $\mathcal{A}' = (Q_{\mathcal{A}'}, \Gamma, \Delta_{\mathcal{A}'}, Q, F_{\mathcal{A}'})$ such that $|\mathcal{A}'| = O(|\mathcal{M}|)$ and $(q'', \bot, \ldots, \bot, w) \in L_{\mathcal{M}}(\mathcal{A})$ if and only if $(q'', \bot, \ldots, \bot, w) \overset{\tau'}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M})} (q, \bot, \ldots, \bot)$ for some $\tau' \in \Delta^*$. Afterwards, we assume w.l.o.g that the $\mathcal{M}$-automaton has no ϵ-transitions.

Let $\mathcal{M}_{[1,n[} = (n, Q, \Gamma, \Delta_{[1,n[})$ be the OMPS built from $\mathcal{M}$ by discarding the set of pop operations of $\mathcal{M}$ over the n^{th} stack. Formally, we have $\Delta_{[1,n[} = \Delta \cap \big((Q \times (\Gamma_\epsilon)^{n-1} \times \{\epsilon\}) \times (Q \times (\Gamma^*)^n)\big)$. Then, it is easy to see that for every configuration $(q', w_1, \ldots, w_n)$ in $Pre^*_{\mathcal{T}(\mathcal{M}')}(\{(q, \bot, \ldots, \bot)\})$, there are $q'' \in Q$, $w \in Stack(\mathcal{M})$, $\tau' \in \Delta^*$, and $\tau \in \Delta^*_{[1,n[}$ such that:

$$(q', w_1, \ldots, w_n) \overset{\tau}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \bot, \ldots, \bot, w) \overset{\tau'}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M})} (q, \bot, \ldots, \bot)$$

Since the OMPS $\mathcal{M}_{[1,n[}$ can only have push operations over its n-th stack, we have $(q', w_1, \ldots, w_n) \overset{\tau}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \bot, \ldots, \bot, w)$ if and only if there is $v \in (\Gamma \setminus \{\bot\})^*$ such that $w = vw_n$ and $(q', w_1, \ldots, w_{n-1}, \bot) \overset{\tau}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \bot, \ldots, \bot, v)$.

On the other hand, let $\mathcal{M}' = (n-1, Q \times Q_{\mathcal{A}'}, \Gamma, \Delta')$ be an $(n-1)$-OMPS built up from the OMPS $\mathcal{M}_{[1,n[}$ and the FSA $\mathcal{A}'$ such that $\langle (q_1, p_1), \gamma_1, \ldots, \gamma_{n-1}\rangle \rightarrow_{\mathcal{M}'} \langle (q_2, p_2), \alpha_1, \ldots, \alpha_{n-1}\rangle$ if and only if $\langle q_1, \gamma_1, \ldots, \gamma_{n-1}, \epsilon\rangle \rightarrow_{\mathcal{M}_{[1,n[}} \langle q_2, \alpha_1, \ldots, \alpha_{n-1}, \alpha_n\rangle$ and $p_2 \overset{\alpha_n}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{A}')} p_1$ for some $\alpha_n \in \big((\Gamma \setminus \{\bot\}) \cup \{\epsilon\}\big)$. In fact, the OMPS $\mathcal{M}'$ defines a kind of synchronous product between the pushed word over the n-th stack of OMPS $\mathcal{M}_{[1,n[}$ and the reverse of the input word of the FSA $\mathcal{A}'$. Observe that the size of the constructed $(n-1)$- OMPS $\mathcal{M}'$ is $O(|\mathcal{M}|^2)$).

Then, the relation between $\mathcal{M}'$, $\mathcal{M}_{[1,n[}$, and $\mathcal{A}'$ is given by Lemma 6 which follows immediately from the definition of $\mathcal{M}'$.

▶ **Lemma 6.** *$((q_1, p_1), w_1, \ldots, w_{n-1}) \overset{\varsigma}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M}')} ((q_2, p_2), \bot, \ldots, \bot)$ if and only if there is a $v \in (\Gamma \setminus \{\bot\})^*$ such that $(q_1, w_1, \ldots, w_{n-1}, \bot) \overset{\tau}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q_2, \bot, \ldots, \bot, v\bot)$ and $p_2 \overset{v}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{A}')} p_1$.*

Now, we can apply the induction hypothesis to $\mathcal{M}'$ to show that for every $(q'', p'') \in Q \times Q_{\mathcal{A}'}$, it possible to construct , in time $O(|\mathcal{M}|^{2^{d(n-1)+2}})$, a $\mathcal{M}'$-automaton $\mathcal{A}_{(q'',p'')}$ such that $|\mathcal{A}_{(q'',p'')}| = O(|\mathcal{M}|^{2^{d(n-1)+2}})$ and $L_{\mathcal{M}'}(\mathcal{A}_{(q'',p'')}) = Pre^*_{\mathcal{T}(\mathcal{M}')}(\{((q'', p''), \bot, \ldots, \bot)\})$.

From the $\mathcal{M}'$-automaton $\mathcal{A}_{(q'',p'')}$ and the $\mathcal{M}$-automaton $\mathcal{A}'$, we can construct a $\mathcal{M}$-automaton $\mathcal{A}$ such that $(q', w_1, \ldots, w_n) \in L_{\mathcal{M}}(\mathcal{A})$ if and only if there are $q'' \in Q$ and $p', p'' \in Q_{\mathcal{A}}$ such that: (1) $q'' \overset{\bot^{n-1}}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{A}')} p''$, (2) $((q', p'), w_1, \ldots, w_{n-1}) \in L_{\mathcal{M}'}(\mathcal{A}_{(q'',p'')})$, and (3) $p' \overset{w_n}{\Longrightarrow}{}^*_{\mathcal{T}(\mathcal{A}')} p$ for some $p \in F_{\mathcal{A}'}$. Observe that such an automaton $\mathcal{A}$ of the size $O(|\mathcal{M}|^{2^{dn}})$ (by taking d as big as needed) is effectively constructible from $\mathcal{A}_{(q'',p'')}$ and $\mathcal{A}'$ using standard automata operations. Moreover, we have:

▶ **Lemma 7.** $L_{\mathcal{M}}(\mathcal{A}) = Pre^*_{\mathcal{T}(\mathcal{M})}(\{(q, \bot, \ldots, \bot)\})$.

Proof. ($\subseteq$) Let $(q', w_1, \ldots, w_n) \in L_{\mathcal{M}}(\mathcal{A})$. Then, there are $q'' \in Q$ and $p', p'' \in Q_{\mathcal{A}}$ such that: (1) $q'' \xRightarrow{\perp^{n-1}}{}^*_{\mathcal{T}(\mathcal{A}')} p''$, (2) $((q', p'), w_1, \ldots, w_{n-1}) \in L_{\mathcal{M}'}(\mathcal{A}_{(q'',p'')})$, and (3) $p' \xRightarrow{w_n}{}^*_{\mathcal{T}(\mathcal{A}')} p$ for some $p \in F_{\mathcal{A}'}$.

So, we can apply Lemma 6 to the run $((q', p'), w_1, \ldots, w_{n-1}) \xRightarrow{\varsigma}{}^*_{\mathcal{T}(\mathcal{M}')} ((q'', p''), \perp, \ldots, \perp)$ to show that there is $v \in \Gamma^*$ such that $(q', w_1, \ldots, w_{n-1}, \perp) \xRightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \perp, \ldots, \perp, v)$ and $p'' \xRightarrow{v}{}^*_{\mathcal{T}(\mathcal{A}')} p'$. Thus, we have $(q', w_1, \ldots, w_{n-1}, w_n) \xRightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M})} (q'', \perp, \ldots, \perp, vw_n)$.

Now, we can use the runs $q'' \xRightarrow{\perp^{n-1}}{}^*_{\mathcal{T}(\mathcal{A}')} p''$, $p'' \xRightarrow{v}{}^*_{\mathcal{T}(\mathcal{A}')} p'$, and $p' \xRightarrow{w_n}{}^*_{\mathcal{T}(\mathcal{A}')} p$ to show that $(q'', \perp, \ldots, \perp, vw_n) \in L_{\mathcal{M}}(\mathcal{A}')$. This implies that $(q'', \perp, \ldots, \perp, vw_n) \xRightarrow{\tau'}{}^*_{\mathcal{T}(\mathcal{M})} (q, \perp, \ldots, \perp)$.

Hence, we have $(q', w_1, \ldots, w_n) \in Pre^*_{\mathcal{T}(\mathcal{M})}(\{(q, \perp, \ldots, \perp)\})$ and therefore $L_{\mathcal{M}}(\mathcal{A}) \subseteq Pre^*_{\mathcal{T}(\mathcal{M})}(\{(q, \perp, \ldots, \perp)\})$.

($\supseteq$) Let $(q', w_1, \ldots, w_n) \in Pre^*_{\mathcal{T}(\mathcal{M})}(\{(q, \perp, \ldots, \perp)\})$. Then, there are $q'' \in Q$, $v \in \Gamma^*$, $\tau' \in \Delta^*$, and $\tau \in \Delta^*_{[1,n[}$ such that:

$$(q', w_1, \ldots, w_n) \xRightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \perp, \ldots, \perp, vw_n) \xRightarrow{\tau'}{}^*_{\mathcal{T}(\mathcal{M})} (q, \perp, \ldots, \perp)$$

Since $(q'', \perp, \ldots, \perp, vw_n) \xRightarrow{\tau'}{}^*_{\mathcal{T}(\mathcal{M})} (q, \perp, \ldots, \perp)$, we have $(q'', \perp, \ldots, \perp, vw_n) \in L_{\mathcal{M}}(\mathcal{A}')$. This implies that there are $p', p'' \in Q_{\mathcal{A}'}$ and $p \in F_{\mathcal{A}'}$ such that $q'' \xRightarrow{\perp^{n-1}}{}^*_{\mathcal{T}(\mathcal{A}')} p''$, $p'' \xRightarrow{v}{}^*_{\mathcal{T}(\mathcal{A}')} p'$, and $p' \xRightarrow{w_n}{}^*_{\mathcal{T}(\mathcal{A}')} p$.

On the other hand, we can show $(q', w_1, \ldots, w_{n-1}, \perp) \xRightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \perp, \ldots, \perp, v)$ since we have $(q', w_1, \ldots, w_n) \xRightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \perp, \ldots, \perp, vw_n)$.

Then, we can apply Lemma 6 to $(q', w_1, \ldots, w_{n-1}, \perp) \xRightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M}_{[1,n[})} (q'', \perp, \ldots, \perp, v)$ and $p'' \xRightarrow{v}{}^*_{\mathcal{T}(\mathcal{A}')} p'$ to show that $((q', p'), w_1, \ldots, w_{n-1}) \xRightarrow{\varsigma}{}^*_{\mathcal{T}(\mathcal{M}')} ((q'', p''), \perp, \ldots, \perp)$. This implies that $((q', p'), w_1, \ldots, w_{n-1}) \in L_{\mathcal{M}'}(\mathcal{A}_{(q'',p'')})$. Now, we can use the definition of the $\mathcal{M}$-automaton $\mathcal{A}$ to show that $(q', w_1, \ldots, w_n) \in L_{\mathcal{M}}(\mathcal{A})$ since we have $q'' \xRightarrow{\perp^{n-1}}{}^*_{\mathcal{T}(\mathcal{A}')} p''$, $((q', p'), w_1, \ldots, w_{n-1}) \in L_{\mathcal{M}'}(\mathcal{A}_{(q'',p'')})$, and $p' \xRightarrow{w_n}{}^*_{\mathcal{T}(\mathcal{A}')} p$ with $p \in F_{\mathcal{A}'}$. Hence, we have $L_{\mathcal{M}}(\mathcal{A}) \supseteq Pre^*_{\mathcal{T}(\mathcal{M})}(\{(q, \perp, \ldots, \perp)\})$.

This terminates the proof of Lemma 7. ◀

This terminates the proof of Theorem 5. ◀

As an immediate consequence of Theorem 5 and Lemma 3, we obtain:

▶ **Theorem 8.** *Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS and $\mathcal{A}'$ be a $\mathcal{M}$-automaton. Then, it is possible to construct, in time $O((|\mathcal{M}| + |\mathcal{A}'|)^{2^{dn}})$ where d is a constant, a $\mathcal{M}$-automaton $\mathcal{A}$ such that $|\mathcal{A}| = O((|\mathcal{M}| + |\mathcal{A}'|)^{2^{dn}})$ and $L_{\mathcal{M}}(\mathcal{A}) = Pre^*_{\mathcal{T}(\mathcal{M})}(L_{\mathcal{M}}(\mathcal{A}'))$.*

We can extend the previous result to show that the operator Pre^+ preservers also recognizability.

▶ **Theorem 9.** *Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS and $\mathcal{A}'$ be a $\mathcal{M}$-automaton. Then, it is possible to construct, in time $O((|\mathcal{M}| + |\mathcal{A}'|)^{2^{dn}})$ where d is a constant, a $\mathcal{M}$-automaton $\mathcal{A}$ such that $|\mathcal{A}| = O((|\mathcal{M}| + |\mathcal{A}'|)^{2^{dn}})$ and $L_{\mathcal{M}}(\mathcal{A}) = Pre^+_{\mathcal{T}(\mathcal{M})}(L_{\mathcal{M}}(\mathcal{A}'))$.*

Proof. For Pre^+, it is sufficient to construct a $\mathcal{M}$-automaton that recognizes the set $Pre_{\mathcal{T}(\mathcal{M})}(L_{\mathcal{M}}(\mathcal{A}))$ which is an easy extension of the construction given in [6] for standard pushdown systems. ◀

5 Applications to Linear-Time Global Model Checking

5.1 The repeated state global reachability problem

Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an ordered multi-pushdown system. In the this section, we are interested in solving *the repeated state global reachability problem* which consists in computing, for a given state $q_f \in Q$, the set of all configurations c of $\mathcal{M}$ such that there is an infinite run of $\mathcal{T}(\mathcal{M})$ starting from c that visits infinitely often the state q_f.

To this aim, let us introduce the following notation: For every $i \in [1, n]$, we denote by $\mathcal{M}_{[1,i]} = (n, Q, \Gamma, \Delta_{[1,i]})$ the OMPS built from $\mathcal{M}$ by discarding pop operations of $\mathcal{M}$ over the last $(n-i)$ stacks. Formally, we have $\Delta_{[1,i]} = \Delta \cap \big((Q \times (\Gamma_\epsilon)^i \times (\{\epsilon\})^{n-i}) \times (Q \times (\Gamma^*)^n)\big)$.

For every $i \in [1, n]$, and every $(q, \gamma) \in Q \times (\Gamma \setminus \{\bot\})$, let $C_i^{(q,\gamma)}$ denote the set of all configurations $(q, w_1, \ldots, w_n) \in \mathit{Conf}(\mathcal{M})$ such that $w_1 = \cdots = w_{i-1} = \bot$ and $w_i = \gamma u$ for some $u \in \mathit{Stack}(\mathcal{M})$. Moreover, let $c_i^{(q,\gamma)}$ be the configuration $(q, w_1, \ldots, w_n)$ of $\mathcal{M}$ such that $w_i = \gamma\bot$ and $w_j = \bot$ for all $j \neq i$. Then, the solution of the *repeated state global reachability problem* is based on the following fact:

▶ **Theorem 10.** *Let c be a configuration of $\mathcal{M}$ and q_f be a state of $\mathcal{M}$. There is an infinite run starting from c that visits infinitely often the state q_f if and only if there is $i \in [1, n]$, $q \in Q$, and $\gamma \in \Gamma$ such that:*

1. $c \in \mathit{Pre}^*_{\mathcal{T}(\mathcal{M})}(C_i^{(q,\gamma)})$, *and*
2. $c_i^{(q,\gamma)} \in \mathit{Pre}^+_{\mathcal{T}(\mathcal{M}_{[1,i]})}\big(\mathit{Pre}^*_{\mathcal{T}(\mathcal{M}_{[1,i]})}(C_i^{(q,\gamma)}) \cap (\{q_f\} \times (\mathit{Stack}(\mathcal{M}))^n)\big)$.

Proof. ($\Rightarrow$) : Let $\rho = c_0 t_0 c_1 t_1 c_2 t_2 \cdots$ be an infinite run of $\mathcal{T}(\mathcal{M})$ starting from $c_0 = c$. Recall that for every $j \in \mathbb{N}$, c_j is a configuration of $\mathcal{M}$ and t_j is a transition of $\mathcal{M}$ such that $c_j \xrightarrow{t_j}_{\mathcal{T}(\mathcal{M})} c_{j+1}$. Let $i \in [1, n]$ be the maximal index such that for every $j \in \mathbb{N}$, there is $k_j \geq j$ such that t_{k_j} is a pop transition over the i-th stack of $\mathcal{M}$. Hence, from the definition of i, there is $r \in \mathbb{N}$ such that for every $h \geq r$, there is $d_h \in [1, i]$ such that the transition t_h is a pop transition over the d_h-th stack of $\mathcal{M}$. This implies that for every $h \geq r$, we have $c_h \xrightarrow{t_h}_{\mathcal{T}(\mathcal{M}_{[1,i]})} c_{h+1}$. Moreover, we must have c_{k_j} is in $Q \times (\{\bot\})^{i-1} \times ((\Gamma \setminus \{\bot\})^* \cdot \mathit{Stack}(\mathcal{M})) \times (\mathit{Stack}(\mathcal{M}))^{n-i}$ since t_{k_j} is a pop operations over the i-th stack of $\mathcal{M}$.

Construct a sequence $\pi = c_{j_0} c_{j_1} c_{j_2} \cdots$ of configurations of $\mathcal{M}$ as follows: c_{j_0} is the first configuration of ρ such that $j_0 \geq r$ and t_{j_0} is a pop transition over the i-stack of $\mathcal{M}$, for every $\ell > 0$, c_{j_ℓ} is the first configuration of ρ such that $j_\ell > j_{\ell-1}$ and t_{j_ℓ} is a pop transition over the i-stack of $\mathcal{M}$. Recall that, by definition, we have for every $l \in \mathbb{N}$, c_{j_l} is in $Q \times (\{\bot\})^{i-1} \times ((\Gamma \setminus \{\bot\})^* \cdot \mathit{Stack}(\mathcal{M})) \times (\mathit{Stack}(\mathcal{M}))^{n-i}$.

Now, for every $l \geq 0$, let $\pi^{(l)}$ be the suffix of π starting at c_{j_l}, and let $m^{(l)}$ be the minimal length of the configurations of $\pi^{(l)}$, where the length of a configuration is defined as the length of its i-th stack.

Construct a subsequence $\pi' = c_{z_0} c_{z_1} c_{z_2} \cdots$ of π as follows: c_{z_0} is the first configuration of π of length $m^{(0)}$; for every $l > 0$, c_{z_l} is the first configuration of $\pi^{(z_{l-1}+1)}$ of length $m^{(z_{l-1}+1)}$.

Since the number of states and stack symbols is finite, there exists a subsequence $\pi'' = c_{x_0} c_{x_1} c_{x_2} \cdots$ of π' whose elements have all the same state q, and the same symbol γ on the top of the i-th stack. Observe that $c_{x_0}, c_{x_1}, c_{x_2}, \ldots$ are in $C_i^{(q,\gamma)}$.

Since ρ is an accepting run, there is an index $b \geq 1$ and a configuration c_{q_f} with state q_f such that:

$$c_0 \xrightarrow{\tau}{}^*_{\mathcal{T}(\mathcal{M})} c_{x_0} \xrightarrow{\tau'}{}^+_{\mathcal{T}(\mathcal{M})} c_{q_f} \xrightarrow{\tau''}{}^*_{\mathcal{T}(\mathcal{M})} c_{x_b}$$

Since $c_0 = c$ and $c_{x_0} \in C_i^{(q,\gamma)}$, we have $c \in Pre^*_{\mathcal{T}(\mathcal{M})}(C_i^{(q,\gamma)})$, and so (1) holds.
Due to the definition of π (and so, π' and π''), we have

$$c_{x_0} \overset{\tau'}{\Longrightarrow}{}^{+}_{\mathcal{T}(\mathcal{M}_{[1,i]})} c_{q_f} \overset{\tau''}{\Longrightarrow}{}^{*}_{\mathcal{T}(\mathcal{M}_{[1,i]})} c_{x_b}$$

Since $c_{x_0} \in Q \times (\{\bot\})^{i-1} \times ((\Gamma \setminus \{\bot\})^* \cdot Stack(\mathcal{M})) \times (Stack(\mathcal{M}))^{n-i}$, then there are $w_i, w_{i+1}, \ldots, w_n \in Stack(\mathcal{M})$ such that $c_{x_0} = (q, \bot, \ldots, \bot, \gamma w_i, w_{i+1}, \ldots, w_n)$. Due to the definition of the subsequence π' and π'' all the configurations of ρ between c_{x_0} and c_{x_b} have a content of the l-th stack (with $i \leq l \leq k$) of the form $w'_l w_l$. In particular, the configuration c_{q_f} is of the form $(q_f, u_1, \ldots, u_{i-1}, u_i w_i, u_{i+1} w_{i+1}, \ldots, u_n w_n)$ and the configuration c_{x_b} is of the form $(q, \bot, \ldots, \bot, \gamma v_i w_i, v_{i+1} w_{i+1}, \ldots, v_n w_n)$. This implies:

$$c_i^{(q,\gamma)} = (q, \bot, \ldots, \bot, \gamma, \bot, \ldots, \bot) \overset{\tau'}{\Longrightarrow}{}^{+}_{\mathcal{T}(\mathcal{M}_{[1,i]})} (q_f, u_1, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots, u_n)$$

and

$$(q_f, u_1, \ldots, u_{i-1}, u_i, u_{i+1}, \ldots, u_n) \overset{\tau''}{\Longrightarrow}{}^{*}_{\mathcal{T}(\mathcal{M}_{[1,i]})} (q, \bot, \ldots, \bot, \gamma v_i, v_{i+1}, \ldots, v_n)$$

Consequently, (2) holds, which concludes the proof.

($\Leftarrow$) : It is easy to see that we can use (1) and (2) to construct a run starting from c that visits infinitely often the state q_f.

◀

Since the sets of configurations $C_i^{(q,\gamma)}$ and $(\{q_f\} \times (Stack(\mathcal{M}))^n)$ are recognizable, we can use Theorem 8 and Theorem 9 to construct $\mathcal{M}$-automata recognizing $Pre^*_{\mathcal{T}(\mathcal{M})}(C_i^{(q,\gamma)})$ and $Pre^+_{\mathcal{T}(\mathcal{M}_{[1,i]})}\big(Pre^*_{\mathcal{T}(\mathcal{M}_{[1,i]})}(C_i^{(q,\gamma)}) \cap (\{q\} \times (Stack(\mathcal{M}))^n)\big)$. Hence, we can construct a $\mathcal{M}$-automaton that recognizes the set of all configurations c of $\mathcal{M}$ such that there is an infinite run of $\mathcal{T}(\mathcal{M})$ starting from c that visits infinitely often the state q_f.

▶ **Theorem 11.** *Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS and $q \in Q$ be a state. Then, it is possible to construct, in time $O((|\mathcal{M}|)^{2^{dn}})$ where d is a constant, a $\mathcal{M}$-automaton $\mathcal{A}$ such that $|\mathcal{A}| = O((|\mathcal{M}|)^{2^{dn}})$ and for every configuration $c \in Conf(\mathcal{M})$, $c \in L_{\mathcal{M}}(\mathcal{A})$ if and only if there is an infinite run of $\mathcal{T}(\mathcal{M})$ starting from c that visits infinitely often the state q.*

5.2 w-regular properties

In this section, we assume that the reader is familiar with w-regular properties expressed in the linear-time temporal logics [18] or the linear time μ-calculus [26]. For more details, the reader is referred to [18, 28, 26, 27].

Let φ be an w-regular formula built from a set of atomic propositions $Prop$, and let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS with a labeling function $\Lambda : Q \to 2^{Prop}$ associating to each state $q \in Q$ the set of atomic propositions that are true in it. In the following, we are interested in solving *the global model checking problem* which consists in computing the set of all configurations c of $\mathcal{M}$ such that every infinite run starting from c satisfies the formula φ.

To solve this problem, we adopt an approach similar to [6, 5] and we construct a Buchi automaton $\mathcal{B}_{\neg\varphi}$ over the alphabet 2^{Prop} accepting the negation of φ [28, 27]. Then, we compute the product of the OMPS $\mathcal{M}$ and of the Büchi automaton $\mathcal{B}_{\neg\varphi}$ to obtain an n-OMPS $\mathcal{M}_{\neg\varphi}$ with a set of repeating states G. Now, it is easy to see that the original problem can be reduced to the *repeated state global reachability problem* which compute the set of all configurations c such that there is an infinite run of $\mathcal{T}(\mathcal{M})$ starting from c that visits infinitely often a state in G. Hence, as an immediate consequence of Theorem 11, we obtain:

▶ **Theorem 12.** *Let $\mathcal{M} = (n, Q, \Gamma, \Delta)$ be an OMPS with a labeling function Λ, and let φ be a linear time μ-calculus formula or linear time temporal formula. Then, it is possible to construct, in time $O((2^{|\varphi|} \cdot |\mathcal{M}|)^{2^{dn}})$ where d is a constant, a $\mathcal{M}$-automaton $\mathcal{A}$ such that $|\mathcal{A}| = O((2^{|\varphi|} \cdot |\mathcal{M}|)^{2^{dn}})$ and for every configuration $c \in Conf(\mathcal{M})$, $c \in L_{\mathcal{M}}(\mathcal{A})$ if and only if there is an infinite run of $\mathcal{T}(\mathcal{M})$ starting from c that does not satisfy φ.*

Proof. It is well known that it is possible to construct, in time exponential in $|\varphi|$, a Büchi automaton $\mathcal{B}_{\neg\varphi}$ for the negation of $\neg\varphi$ having exponential size in $|\varphi|$ [28, 26]. Therefore, the product of $\mathcal{M}$ and $\mathcal{B}_{\neg\varphi}$ has polynomial size in $|\mathcal{M}|$ and exponential size in $|\varphi|$. Applying Theorem 11 to the n-OMPS $\mathcal{M}_{\neg\varphi}$ (the product of $\mathcal{M}$ and $\mathcal{B}_{\neg\varphi}$) of size $O(2^{|\varphi|} \cdot |\mathcal{M}|)$ we obtain our complexity result. ◀

Observe that we can also construct a $\mathcal{M}$-automaton $\mathcal{A}'$ such that for every configuration $c \in Conf(\mathcal{M})$, $c \in L_{\mathcal{M}}(\mathcal{A})$ if and only if every infinite run of $\mathcal{T}(\mathcal{M})$ starting from c that satisfies φ since the class of $\mathcal{M}$-automata is closed under boolean operations.

We are now ready to establish our result about the model checking problem for w-regular properties which consists in checking whether, for a given configuration c of the OMPS, every infinite run starting from c satisfies the formula φ.

▶ **Theorem 13.** *The model checking problem for the linear-time temporal logics or the linear-time μ-calculus and OMPSs is 2ETIME-complete.*

Proof. The 2ETIME upper bound is established by Theorem 12. To prove hardness, we use the fact that the emptiness problem for ordered multi-pushdown automata is 2ETIME-complete [1]. ◀

6 Conclusion

We have shown that the set of all predecessors of a recognizable set of configurations of an ordered multi-pushdown system is an effectively constructible recognizable set. We have also proved that the set of all configurations of an ordered multi-pushdown system that satisfy a given w-regular property is effectively recognizable. From these results we have derived an 2ETIME upper bound for the model checking problem of w-regular properties.

It may be interesting to see if our approach can be extended to solve the global model-checking problem for branching time properties expressed in CTL or CTL* by adapting the constructions given in [5, 12] for standard pushdown systems.

Acknowledgements I want to thank Ahmed Bouajjani who greatly helped by reading this paper at various stages.

References

1 M. F. Atig, B. Bollig, and P. Habermehl. Emptiness of multi-pushdown automata is 2ETIME-complete. In *Proceedings of DLT'08*, volume 5257 of *LNCS*, pages 121–133. Springer, 2008.

2 M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR*, volume 5201 of *LNCS*, pages 356–371. Springer, 2008.

3 M. F. Atig and T. Touili. Verifying parallel programs with dynamic communication structures. In *CIAA*, volume 5642 of *LNCS*, pages 145–154. Springer, 2009.

4 Mohamed Faouzi Atig. From multi to single stack automata. In *CONCUR*, volume 6269 of *Lecture Notes in Computer Science*, pages 117–131. Springer, 2010.

5 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, volume 1243 of *LNCS*, pages 135–150. Springer, 1997.

6 A. Bouajjani and O. Maler. Reachability analysis of pushdown automata. In *Proc. Intern. Workshop on Verification of Infinite-State Systems (Infinity'96)*, 1996.

7 A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR'05*, LNCS, 2005.

8 A. Bouajjani and T. Touili. Reachability Analysis of Process Rewrite Systems. In *FSTTCS'03*. LNCS 2914, 2003.

9 L. Breveglieri, A. Cherubini, C. Citrini, and S. Crespi Reghizzi. Multi-push-down languages and grammars. *International Journal of Foundations of Computer Science*, 7(3):253–292, 1996.

10 J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *FoSSaCS*, volume 1578 of *LNCS*, pages 14–30. Springer, 1999.

11 J. Esparza and A. Podelski. Efficient algorithms for pre* and post* on interprocedural parallel flow graphs. In *POPL'00*. ACM, 2000.

12 Alain Finkel, Bernard Willems, and Pierre Wolper. A direct symbolic approach to model checking pushdown systems (extended abstract). In Faron Moller, editor, *Proceedings of the 2nd International Workshop on Verification of Infinite State Systems (INFINITY'97)*, volume 9 of *Electronic Notes in Theoretical Computer Science*, pages 27–39, Bologna, Italy, July 1997. Elsevier Science Publishers.

13 Alexander Heußner, Jérôme Leroux, Anca Muscholl, and Grégoire Sutre. Reachability analysis of communicating pushdown systems. In *FOSSACS*, volume 6014 of *Lecture Notes in Computer Science*, pages 267–281. Springer, 2010.

14 Ranjit Jhala and Rupak Majumdar. Interprocedural analysis of asynchronous programs. In *POPL*. IEEE, 2007.

15 V. Kahlon. Boundedness vs. unboundedness of lock chains: Characterizing decidability of pairwise cfl-reachability for threads communicating via locks. In *LICS*, pages 27–36. IEEE Computer Society, 2009.

16 A. Lal and T.W. Reps. Reducing concurrent analysis under a context bound to sequential analysis. In *CAV*, volume 5123 of *LNCS*, pages 37–51. Springer, 2008.

17 D. Lugiez and Ph. Schnoebelen. The regular viewpoint on pa-processes. In *CONCUR*, volume 1466 of *LNCS*, pages 50–66. Springer, 1998.

18 Amir Pnueli. The temporal logic of programs. In *FOCS*, pages 46–57. IEEE, 1977.

19 S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, volume 3440 of *LNCS*, pages 93–107. Springer, 2005.

20 G. Ramalingam. Context-sensitive synchronization-sensitive analysis is undecidable. *ACM Trans. Program. Lang. Syst.*, 22(2):416–430, 2000.

21 T.W. Reps, S. Schwoon, and S. Jha. Weighted pushdown systems and their application to interprocedural dataflow analysis. In *SAS*, volume 2694 of *LNCS*, pages 189–213. Springer, 2003.

22 K. Sen and M. Viswanathan. Model checking multithreaded programs with asynchronous atomic methods. In *CAV*, pages 300–314. LNCS 4144, 2006.

23 Anil Seth. Global reachability in bounded phase multi-stack pushdown systems. In *CAV*, volume 6174 of *Lecture Notes in Computer Science*, pages 615–628. Springer, 2010.

24 S. La Torre, P. Madhusudan, and G. Parlato. A robust class of context-sensitive languages. In *Proceedings of LICS*, pages 161–170. IEEE, 2007.

25 S. La Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *Proceedings of TACAS'08*, LNCS, pages 299–314. Springer, 2008.

26 Moshe Y. Vardi. A temporal fixpoint calculus. In *POPL*, pages 250–259, 1988.

27 Moshe Y. Vardi. Alternating automata and program verification. In *Computer Science Today*, volume 1000 of *Lecture Notes in Computer Science*, pages 471–485. Springer, 1995.

28 Moshe Y. Vardi and Pierre Wolper. An automata-theoretic approach to automatic program verification (preliminary report). In *LICS*, pages 332–344. IEEE Computer Society, 1986.

The Complexity of Model Checking (Collapsible) Higher-Order Pushdown Systems

Matthew Hague[1] and Anthony Widjaja To[2]

1,2 Oxford University Computing Laboratory
Wolfson Building, Parks Road
Oxford, OX1 3QD

Abstract

We study (collapsible) higher-order pushdown systems — theoretically robust and well-studied models of higher-order programs — along with their natural subclass called (collapsible) higher-order basic process algebras. We provide a comprehensive analysis of the model checking complexity of a range of both branching-time and linear-time temporal logics. We obtain tight bounds on data, expression, and combined-complexity for both (collapsible) higher-order pushdown systems and (collapsible) higher-order basic process algebra. At order-k, results range from polynomial to $(k+1)$-exponential time. Finally, we study (collapsible) higher-order basic process algebras as graph generators and show that they are almost as powerful as (collapsible) higher-order pushdown systems up to MSO interpretations.

1998 ACM Subject Classification D.2.4

Keywords and phrases Higher-Order, Collapsible, Pushdown Systems, Temporal Logics, Complexity, Model Checking

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.228

1 Introduction

Recently, there has been a burgeoning interest in collapsible higher-order pushdown systems (CPDSs), both as generators of structures and as models of higher-order computation. Whereas an order-1 pushdown system augments a finite-state automaton with an unbounded stack memory, a higher-order pushdown system (HOPDS) provides a nested "stack-of-stacks" structure. CPDSs allow a further backtracking operation called *collapse*.

Higher-order pushdown automata (HOPDA) were introduced by Maslov [24]. *Higher-order pushdown systems* (HOPDS) are HOPDA viewed as generators of infinite trees or graphs. Recently these models have been generalised to *collapsible pushdown systems* (CPDS) [17, 19]. In terms of expressivity, order-k CPDSs generate the same class of ranked trees as deterministic order-k recursion schemes [17]. The analogous result holds for *safe* recursion schemes and HOPDSs [18]. These systems provide a natural model for higher-order programs with (unbounded) recursive function calls and are therefore useful in software verification. Further results show an intimate connection with the Caucal hierarchy [10, 11]. For verification, reachability properties — which ask whether a given set of control states can be reached from the initial configuration — are complete for $(k-1)$-ExpTime [5, see appendix], whilst μ-calculus properties are k-ExpTime-complete [7, 26, 17]. Despite these high complexities, Kobayashi has verified resource usage properties of higher-order programs [20] using a novel approach based on intersection types [21, 23].

Hitherto, there has been little work addressing the precise complexity of model checking higher-order programs with respect to the common temporal logics. In most cases, there

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 228–239

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

	(Collapsible) HOPDS		(Collapsible) HOBPA	
	Data	Expression & Combined	Data	Expression & Combined
μLTL / LTL	$(k-1)$-ExpTime	k-ExpTime	P-time	k-ExpTime
LTL(F, X)	$(k-1)$-ExpTime	k-ExpTime	P-time	k-ExpTime
LTL(U)	$(k-1)$-ExpTime	k-ExpTime	P-time	k-ExpTime
CTL	k-ExpTime	k-ExpTime	P-time	k-ExpTime
CTL+	k-ExpTime	$(k+1)$-ExpTime	P-time	$(k+1)$-ExpTime
CTL*	k-ExpTime	$(k+1)$-ExpTime	P-time	$(k+1)$-ExpTime
EF	$(k-1)$-ExpSpace-hard	$(k-1)$-ExpSpace-hard	P-time	$(k-1)$-ExpSpace-hard

Figure 1 The complexity of model checking order-k higher-order systems. Unless stated, all results are complete.

is currently a single or double exponential gap in the best known upper and lower bounds (derived usually from μ-calculus and reachability respectively). One main contribution of this paper is a nearly complete picture of the model checking complexities against temporal logics. In particular, we consider data complexity (formulas are fixed), expression complexity (systems are fixed), and combined complexity (both formulas and systems are input parameters). Table 1 (left column) summarises our results. In all cases, our lower bounds hold without the collapse operation, whilst our upper bounds allow collapse.

Basic process algebras (BPAs) are a natural and well-studied subclass of order-1 PDSs (cf. [6]), which are suitable abstractions for modelling the control-flow of sequential programs (cf. [2, 14]). We propose higher-order extensions of BPAs, called *(collapsible) higher-order basic process algebras* (HOBPAs), that form a natural subclass of (collapsible) HOPDSs. This differs from the single-state HOPDSs introduced by Bouajjani and Meyer [3]. As graph generators, (collapsible) HOBPAs are almost as powerful as (collapsible) HOPDSs in the following sense: (1) like CPDS, there exists a collapsible order-2 BPA whose graph has an undecidable monadic second-order logic (MSO) theory, and (2) the class of graphs generated by order-k BPAs coincide with those generated by order-k PDSs up to MSO interpretations. In this paper, we provide an almost complete picture for the model checking complexities of standard temporal logics over (collapsible) HOBPAs. See Table 1 (right column). We show that the restriction to HOBPA does not, in most cases, simplify the model checking problem; a notable exception is for data complexity, where the problem becomes polynomial time. Again, our lower bounds hold without collapse, whilst our upper bounds allow collapse.

Similar analyses appear across a number of papers for the special case of order-1 pushdown systems [1, 6, 30, 31, 4]. In all cases we generalise the resulting picture in a natural manner. That is, a 1-ExpTime-complete complexity becomes k-ExpTime-complete, and so on. Our upper bound results concern the data complexity of Collapsible HOBPAs and the data and combined complexities for LTL over CPDSs. Previous work studied reachability, LTL and the alternation-free μ-calculus [16, 27, 13] over HOPDS without collapse. However, we believe the LTL algorithm contains an error, and provide a new algorithm. Furthermore, the alternation-free μ-calculus algorithm in [16] is not optimal. Our remaining results concern lower bounds. We begin with two techniques from in the literature: (1) Engelfriet's characterization of complexity classes k-ExpTime by extensions of HOPDAs (e.g. with space-bounded worktape) [13], and (2) Cachat and Walukiewicz's more "direct" approach via encodings of large numbers using HOPDSs [8]. We employ Technique (1) to prove the lower bounds for LTL (and its fragments), CTL, CTL+, and CTL*. This does not mean that the proofs of the results are immediate: it was left as an open problem in [8] whether the two techniques can be used to derive these lower bounds. Since Technique (1) seems only suited to deriving k-ExpTime lower bounds (for some k), we give two variations of

Technique (2) to derive $(k-1)$-ExpSpace lower bounds for EF model checking over HOPDSs and HOBPAs (the latter proof is substantially more involved). The lower bound proofs in this paper suggest that Technique (1) yields simpler proofs, while Technique (2) offers more flexibility.

The preliminaries are given in §2. We begin in §3 with the results for fixed formulas over collapsible HOBPA. In §4 we discuss branching-time logics, and linear-time in §5. Finally, we conclude this paper with future work in §6. Due to the length and intricate nature of the proofs, we relegate the full details into the full version.

2 Preliminaries

We define (collapsible) higher-order pushdown systems and basic process algebra and give a result of Engelfriet used in some proofs. Note, after defining higher-order and collapsible stores, we only define higher-order systems. For the collapsible version, simply replace the higher-order store with a collapsible one, expanding the stack operations accordingly. Also, the definitions generalise from non-deterministic to alternating in the standard way.

Higher-Order Collapsible Pushdown Stores

We begin by defining a higher-order pushdown store. Collapse links will be introduced afterwards. Intuitively, a higher-order store is a stack of lower order stacks.

▶ **Definition 1** (k-Stores). Let C_0^Σ be a finite alphabet Σ with $[,] \notin \Sigma$. For $k \geq 1$, the set of *k-stores* C_k^Σ contains all $[\gamma_1 \dots \gamma_m]$ with $m \geq 1$ and $\gamma_i \in C_{k-1}^\Sigma$ for all $1 \leq i \leq m$.

There are two operations defined over 1-stores (for all $w \in \Sigma^*$)

$$push_w[a_1 \dots a_m] = [wa_2 \dots a_m] \quad \text{and} \quad top_1[a_1 \dots a_m] = a_1 \ .$$

We define $pop_1 = push_\varepsilon$. Let $\mathcal{O}_1 = \{\ push_w \mid w \in \Sigma^*\ \}$. When $k > 1$, a push operation creates a copy of the topmost stack, while a pop removes it. We assume w.l.o.g. that $\Sigma \cap \mathbb{N} = \emptyset$, where $\mathbb{N}$ is the set of natural numbers. Finally, let $[\gamma_1 \dots \gamma_m] \in C_k^\Sigma$ for some k.

$$\begin{array}{rcll}
push_w[\gamma_1 \dots \gamma_m] & = & [push_w(\gamma_1)\gamma_2 \dots \gamma_m] & \\
push_l[\gamma_1 \dots \gamma_m] & = & [push_l(\gamma_1)\gamma_2 \dots \gamma_m] & \text{if } 2 \leq l < k \\
push_k[\gamma_1 \dots \gamma_m] & = & [\gamma_1\gamma_1\gamma_2 \dots \gamma_m] & \\
pop_l[\gamma_1 \dots \gamma_m] & = & [pop_l(\gamma_1)\gamma_2 \dots \gamma_m] & \text{if } 1 \leq l < k \\
pop_k[\gamma_1 \dots \gamma_m] & = & [\gamma_2 \dots \gamma_m] & \text{if } m > 1 \\
top_l[\gamma_1 \dots \gamma_m] & = & top_l(\gamma_1) & \text{if } 1 \leq l < k \\
top_k[\gamma_1 \dots \gamma_m] & = & \gamma_1 &
\end{array}$$

Note, when $m = 1$, pop_k is undefined. Let $\mathcal{O}_k = \{\ push_w \mid w \in \Sigma^*\ \} \cup \{\ push_l, pop_l \mid 1 < l \leq k\ \}$. We designate $\perp$ to be a *bottom of stack* symbol that is neither pushed nor popped. Let $[w]_1 = [w]$ and $[w]_k = [[w]_{k-1}]$.

For collapse, the order-1 push operation $push_w$ is replaced with $push_{a_1^{i_1} \dots a_m^{i_m} b}$ for $1 \leq i_z \leq k$ and $a_z, b \in \Sigma$ where $1 \leq z \leq m$. A $push_{a_1^{i_1} \dots a_m^{i_m} b}$ on some stack with top_1 character a is equivalent to $push_{a_1 \dots a_m b}$ except each a_z is augmented with a pair $(i_z, 1)$. That is, the top of stack character $a^{(i,j)}$ is replaced by $a_1^{(i_1,1)} \dots a_m^{(i_m,1)} b^{(i,j)}$. The collapse operation from a character $a^{(i,j)}$ is equivalent to j applications of pop_i. The second component j is incremented at every $push_i$. Hence, (i, j) is a link to the order-$(i-1)$ stack beneath the character when it was first pushed.

Consider $[[[\bot]\,[\bot]]]$. Applying $push_{a^2\bot}$ gives $\left[\left[\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\right]$. The pair $(2,1)$ points to $[\bot]$. A $push_2$ leads to $\left[\left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\right]$. Note the second component is incremented in the copy of a, and, thus, $(2,2)$ also points to $[\bot]$. A subtlety occurs after a $push_3$. We obtain the stack below on the left, where the copies of a now refer to the copies of $[\bot]$ within the order-2 stack they occupy. After a collapse, we obtain the stack on the right.

$$\left[\begin{array}{l}\left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\\ \left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\end{array}\right] \qquad \left[\begin{array}{c}[[\bot]]\\ \left[\left[a^{(2,2)}\ \bot\right]\,\left[a^{(2,1)}\ \bot\right]\,[\bot]\right]\end{array}\right]$$

Formally, we define order-k stores with links in terms of order-k stores over the infinite alphabet $\Sigma = \{\ a^{(i,j)}\ \mid i,j \in \mathbb{N}\ \}$. The set of operations over an order-k store with links is

$$\begin{aligned}\mathcal{O}_k^c \quad = \quad & \left\{\ push_{a_1^{i_1}\ldots a_m^{i_m} b}\ \mid \forall 1 \le z \le m . 1 \le i_z \le k \wedge a_z \in \Sigma\ \right\}\\ & \cup \{\ push_l, pop_l, collapse\ \mid 1 < l \le k\ \}\ .\end{aligned}$$

Note that this set of operations is slightly different from the original definition [17]. We show, in the full version, that the definitions are equivalent. The semantics of the operations are given below, in terms of the standard order-k pushdown operators, and an order-k stack $\gamma = [\gamma_1 \ldots \gamma_m]$. Let $\gamma^{<k>}$ be the stack γ where each superscript (i,j) with $i \ge k$ is replaced with $(i, j+1)$.

$$\begin{array}{rcll} push_{a_1^{i_1}\ldots a_m^{i_m} b}(\gamma) & = & push_{a_1^{(i_1,1)}\ldots a_m^{(i_m,1)} b_m^{(i',j')}}(\gamma) & \text{where } top_1(\gamma) = b^{(i',j')}\\ collapse(\gamma) & = & pop_i^j(\gamma) & \text{where } top_1(\gamma) = b^{(i,j)}\\ push_k[\gamma_1 \ldots \gamma_m] & = & [\gamma_1^{<k>}\gamma_1 \ldots \gamma_m] & \\ push_l[\gamma_1 \ldots \gamma_m] & = & [push_l(\gamma_1)\gamma_2 \ldots \gamma_m] & \text{where } l < k \end{array}$$

Higher-Order Pushdown Systems

A HOPDS is a finite-state system with a higher-order store. The finite-state component is the control state. At each step, the applicable transitions are determined by the control state and the top_1 character of the stack. Each transition updates the control state and the stack.

▶ **Definition 2.** An *order-k PDS* is a tuple $(P, \mathcal{R}, \Sigma, p_0, \bot)$ where P is a finite set of control states, $\mathcal{R} \subseteq P \times \Sigma \times \mathcal{O}_k \times P$ is a finite set of rules, Σ is a finite stack alphabet, $p_0 \in P$ is an initial control state and $\bot \in \Sigma$ is a bottom of stack symbol.

A configuration of a higher-order PDS is a pair $\langle p, \gamma\rangle$ where $p \in P$ and γ is a k-store. We have a transition $\langle p, \gamma\rangle \hookrightarrow \langle p', \gamma'\rangle$ iff we have $(p, a, o, p') \in \mathcal{R}$, $top_1(\gamma) = a$ and $\gamma' = o(\gamma)$. The initial configuration is $\langle p_0, [\bot]_k\rangle$.

Higher-Order Basic Process Algebra

An order-1 BPA is an order-1 PDS with a single control state. By applying the same restriction, Bouajjani and Meyer have obtained one definition of higher-order BPA [3]. However, consider $\langle p, [[a\ \bot]]\rangle$ and the rule (omitting the control) $(a, push_2)$. We obtain $\langle p, [[a\ \bot]\,[a\ \bot]]\rangle$ and the same rule can be applied, ad infinitum. However, at order-1 we may use $(a, push_{bc})$ to rewrite the top character before adding a new top character. Hence, at order-j, it is natural to be able to rewrite the top order-$(j-1)$ stack, before adding a new one. Consequently, we introduce $(a, push_j, b) = push_a; push_j; push_b$ for all $2 \le j \le k$. E.g., such rules can simulate $push_2; push_3; pop_2$. Let $\mathcal{O}'_k = \{\ push_w \mid w \in \Sigma^*\ \} \cup \{\ (a, push_j, b), pop_j \mid 1 < j \le k\ \}$.

▶ **Definition 3.** An *order-k BPA* is a tuple $(\mathcal{R}, \Sigma, \bot)$ where $\mathcal{R} \subseteq \Sigma \times \mathcal{O}'_k$ is a finite set of rules, Σ is a finite stack alphabet, and $\bot \in \Sigma$ is the bottom of stack symbol.

We mention two results on the expressive power of HOBPAs as graph generators. Familiarity with monadic second-order logic (MSO) is assumed (cf. [28]). As graph generators, (collapsible) HOBPAs are as powerful as (collapsible) HOPDAs up to monadic second-order logic (MSO) interpretation in the following sense. First, it is known that there exists an order-2 CPDA generating a graph with an undecidable MSO theory [17]. In contrast, over HOPDAs, MSO is decidable. This CPDA is not a collapsible HOBPA. On the other hand, using the ideas from [17], it is not difficult to come up with an order-2 collapsible HOBPA generating a graph with an undecidable MSO theory.

▶ **Proposition** 1. There exists a fixed collapsible order-2 BPA which generates a graph with an undecidable MSO theory.

We sketch the proof of this proposition in the full version. Secondly, we discuss the expressive power of HOBPAs without collapse. Carayol and Wöhrle [9, 10] gave a fixed graph Δ_2^k, for each integer $k > 0$, such that the class of graphs that are MSO-interpretable in the graphs generated by order-k PDSs coincide with the class of graphs that are MSO-interpretable in Δ_2^k. It is easy to check that Δ_2^k can be generated by a fixed order-k BPAs (e.g. see [9]), which implies the following proposition.

▶ **Proposition** 2. The class of graphs that are MSO-interpretable in the graphs generated by order-k BPAs coincide with the class of graphs MSO-interpretable in the graphs generated by order-k PDSs.

Higher-Order Pushdown Automata with an Auxiliary Work Tape

For the lower bound proofs we use HOPDA with a space-bounded work tape. That is, in addition to the control state and the stack, the machine has a bounded, two-way work tape. This tape operates identically to the tape in a Turing machine.

▶ **Definition 4.** An *order-k PDA with an $s(n)$-space work tape* is a tuple $(P, \mathcal{R}, \Sigma, \Gamma \cup \{\varepsilon\}, \Delta, p_0, \bot, \square, \mathcal{F})$ where P is a finite set of control states, $\mathcal{R} \subseteq (P \times \Gamma \cup \{\varepsilon\} \times \Sigma \times \Delta) \times \mathcal{O}_k \times (\Delta \times \{l, r\} \times P)$ is a finite set of rules, Σ is a finite stack alphabet, Γ is a finite input alphabet, Δ is a finite tape alphabet, $p_0 \in P$ is an initial control state, $\bot \in \Sigma$ is the bottom of stack symbol, $\square \in \Delta$ denotes a blank tape cell and $\mathcal{F} \subseteq P$ is a set of accepting control states.

Given an input word of length n, a configuration of a HOPDA with $s(n)$ bounded work tape is a tuple $\langle p, \gamma, t, j\rangle$ where $p \in P$, γ is a k-store, t (the tape contents) is a word in $\Delta^{s(n)}$ and $1 \leq j \leq s(n)$ indicates the position of the read/write head on the tape.

A rule $(p, \alpha, a, x, o, y, d, p') \in \mathcal{R}$ can be applied when the current control state is p, the input character is α, the top-of-stack character is a, and the tape contents at position j are x. The control state is then updated to p', the command o is applied to the stack, and y is written to the tape. The tape head moves accordingly for $d = l$ (left) or $d = r$ (right).

More formally, we have a transition $\langle p, \gamma, t, j\rangle \overset{\alpha}{\hookrightarrow} \langle p', \gamma', t', j'\rangle$ iff we have $(p, \alpha, a, x, o, y, l, p') \in \mathcal{R}$, $j > 1$, $top_1(\gamma) = a$, $t(j) = x$, $\gamma' = o(\gamma)$, $t'(j) = y$, $t'(h) = t(h)$ for all $h \neq j$ and $j' = j - 1$ or we have $(p, \alpha, a, x, o, y, r, p') \in \mathcal{R}$, $j < s(n)$, $top_1(\gamma) = a$, $t(j) = x$, $\gamma' = o(\gamma)$, $t'(j) = y$, $t'(h) = t(h)$ for all $h \neq j$ and $j' = j - 1$. For $\alpha \neq \varepsilon$, we write $c \overset{\alpha}{\hookrightarrow}_\varepsilon c'$ whenever there is a sequence of ε-transitions from c to some c_1, an α-transition from c_1 to c_2 and a sequence of ε-transitions to c'. A word $\alpha_1, \ldots, \alpha_n$ is accepted by the automaton iff $c_n = \langle p, \gamma\rangle$ and $p \in \mathcal{F}$ and $c_0 \overset{\alpha_1}{\hookrightarrow}_\varepsilon \cdots \overset{\alpha_n}{\hookrightarrow}_\varepsilon c_n$ where $c_0 = \langle p_0, [\bot]_k, \square^{s(n)}, 1\rangle$.

Temporal Logics

We will assume familiarity with the temporal logics discussed, remarking only that μLTL is LTL extended with fixed point operators. Full definitions can be found in the literature [12, 29]. We assume, for all logics, the valuations of atomic propositions depend only on the control state and current top-of-stack character, referred to as a *head*. That is, $\Lambda : P \times \Sigma \to 2^{Prop}$ is an assignment of satisfied atomic propositions from the set $Prop$ to each head in $P \times \Sigma$. We say a system satisfies a formula if it holds at the initial state of the system.

Engelfriet's Results

We use the following theorem of Engelfriet [13] in some proofs. Let NSPACE($s(n)$)-P^k denote the class of languages accepted by a non-deterministic order-k PDA with an $s(n)$-space-bounded work tape, where n is the length of the input word. Similarly, ASPACE($s(n)$)-P^k denotes the class of languages accepted by an alternating order-k PDA with an $s(n)$-space-bounded work tape. Finally $\bigcup_{d>0}$ DTIME($exp_k(ds(n))$) is the class of languages accepted by a time-bounded Turing machine, where $exp_0(x) = x$ and $exp_k(x) = 2^{exp_{k-1}(x)}$.

▶ **Theorem 5** ([13], Thm. 2.5). *For any $k \geq 1$ and $s(n) \geq log(n)$, we have NSPACE($s(n)$)-P^k = ASPACE($s(n)$)-P^{k-1} = $\bigcup_{d>0}$ DTIME($exp_k(ds(n))$).*

That is, a non-deterministic order-k PDA with a polynomially-bounded work tape exists for every k-ExpTime language, and an alternating order-k PDA with a polynomially-bounded work tape exists for every $(k+1)$-ExpTime language.

3 Model Checking Collapsible HOBPA Against Fixed Formulas

We begin with a P-time algorithm for model checking collapsible HOBPA against fixed formulas. Hardness follows from the P-time-hardness of context-free language emptiness [15].

▶ **Theorem 6.** *For any logic that can be translated into μ-calculus, model checking collapsible HOBPA against a fixed formula is in P-time.*

Proof. As argued in the full version, any collapsible HOBPA can be simulated by a CPDS with a fixed number of control states. Therefrom, and since the formula is fixed, we construct a CPDS parity game with a fixed number of control states. At order-k, the winner of these games can be determined in k-ExpTime in the number of control states, and polynomial in the alphabet [17]. Hence, the algorithm runs in P-time. ◀

4 Branching Time

We begin by observing, for CPDS, the upper bounds for CTL, CTL+ and CTL* can be obtained by translating into μ-calculus, which has a k-ExpTime model checking problem. For CTL, the translation is polynomial. For CTL+ and CTL* it is exponential, giving $(k+1)$-ExpTime, and k-ExpTime when the formula is fixed. For the lower bound results, we discuss EF, CTL and then CTL+.

▶ **Theorem 7.** *For a fixed formula, and a given order-k CPDS, model checking CTL, CTL+ and CTL* is in k-ExpTime. For a non-fixed system and non-fixed formula, CTL is k-ExpTime, and CTL+ and CTL* are in $(k+1)$-ExpTime.*

4.1 Lower Bounds for EF

In most cases, we are able to derive optimal lower bounds using Theorem 5. However, Theorem 5 is not immediately applicable for (e.g.) $(k-1)$-ExpSpace problems. In the case of EF-logic, the model checking problem over order-1 PDSs and BPAs is PSpace-complete [25, 31]. We now give $(k-1)$-ExpSpace lower bounds for data complexity of order-k PDSs and the expression complexity of order-k BPAs (and thus of order-k PDSs) using the technique of [8] of encoding large numbers. We conjecture that these lower bounds are tight (currently, the best upper bound is k-ExpTime, which is inherited from μ-calculus).

▶ **Theorem 8.** *Model checking EF over order-k PDS without collapse is $(k-1)$-ExpSpace-hard, even for a fixed formula.*

Proof. (sketch) We reduce membership for a given $(k-1)$-ExpSpace Turing machine M using $exp_{k-1}(p(n))$ space on an input word of length n, for some polynomial function p. Fix a number $m \in \mathbb{Z}_{>0}$, which we will later define as $p(n)$ once n is set. The proof combines the technique of [1] for proving that EF-logic over PDS is PSpace-hard and the technique of [8] for encoding and checking large numbers (i.e. k-towers of exponentials) using operations in $\mathcal{O}_k$.

We shall start by briefly recalling the encoding techniques of large numbers from [8]. For each $i \in \mathbb{Z}_{>0}$, we define $\Sigma_i := \{a_i, b_i\}$ and $\Sigma_{\leq i} := \bigcup_{j=1}^{i} \Sigma_i$. We now define the notion of *i-counters* by induction. A *1-counter* (of length m) is a word $\sigma_{m-1} \dots \sigma_0 \in (\Sigma_1)^m$. Such a word naturally represents the number $\sum_{i=0}^{m-1} \sigma_i 2^i$ where a_1 represents 0 and b_1 represents 1. Assuming that the notion of i-counter has been defined, an *$(i+1)$-counter* is simply a word $\sigma_r l_r \dots \sigma_0 l_0$ over $\Sigma_{\leq i+1}$, where $r = exp_i(m) - 1$, $\sigma_j \in \Sigma_{i+1}$, and l_j is an i-counter representing the number j. This $(i+1)$-counter represents the number $\sum_{j=0}^{r} \sigma_j 2^j$, where (as before) a_{i+1} and b_{i+1} are used to (respectively) represent 0 and 1.

Cachat and Walukiewicz [8] showed that a polynomial-size order-k pushdown game arena $\mathcal{P}$ with a reachability objective could be defined (depending only on m) with the following control states and properties: $counter_k$ — from configuration $(counter_k, \gamma)$ of $\mathcal{P}$, Player 0 wins iff γ ends with a k-counter; $first_k$ (resp. $last_k$) — from configuration $(first_k, \gamma)$ (resp. $(last_k, \gamma)$), Player 0 wins iff γ ends with a k-counter representing 0 (resp. $exp_{k-1}(m)$); $equal_k$ — from $(equal_k, \gamma)$, Player 0 wins iff γ ends with two k-counters representing equal values; $succ_k$ — from $(succ_k, \gamma)$, Player 0 wins iff γ ends with two k-counters representing successive values. We observe that the game element of $\mathcal{P}$ can easily be translated into fixed EF formulas (i.e. not depending on m) satisfying the same properties, the main reason being that the game arena $\mathcal{P}$ has a fixed number of rounds.

The rest of the proof uses the idea of [1]. Using an EF operator, we will first guess a word in $\Sigma_{\leq k+1}$ representing an accepting computation of M on the given input word $w = \alpha_1 \dots \alpha_n$. We then need to check that the guess is valid. That is, it represents a sequence of configurations, the initial configuration is the right form, the final configuration is reached, and consecutive configurations respect the transition relation. All these can be done by means of a fixed formula, thanks to the result above for encoding large numbers. ◀

▶ **Theorem 9.** *For a fixed order-k HOBPA without collapse, model checking EF is $(k-1)$-ExpSpace-hard.*

Proof. (sketch) The proof uses some general ideas from the previous proof, but, without control states to encode tests for large numbers, we need an entirely different construction. We briefly explain the order-2 case. Our HOBPA $\mathcal{P}$ will guess an accepting run of a fixed exponential space Turing machine M accepting an ExpSpace-complete language, obtaining a

stack of the form $[w]_2$. For the checking stage, our HOBPA $\mathcal{P}$ now tries to find some location inside the stack that is invalid. In doing so, we need to ensure that all of the information on top of this location is not destroyed. To this end, we will build a stair-like structure from $[w]_2$ by performing operations of the form $[push_1(a'); push_2; push_1(prime)]$ or of the form $[push_1(a''); push_2; push_1(dprime)]$ when seeing a topmost stack symbol a. Here, *prime* and *dprime* are simply intermediate symbols to help signify the action that was previous executed, i.e., we could simply only allow pop_1 operation when *prime* or *dprime* is seen as topmost symbol. The double prime marking is used to "remember" the *starting* point of (sub)configuration that we suspect is invalid. That is, we will have to make sure that it is put precisely once. At some point, $\mathcal{P}$ simply applies rules of the form $push_1(a')$ when a is seen without applying $push_2$, which marks the *end* point of a (sub)configuration that we suspect is invalid. We then only allow rules pop_2 when primed or double primed symbols are seen. To make sure that we see precisely one separator symbol (i.e. $a_3 \in \Sigma_3$), we can use an EF formula saying facts about the location of the double primed symbol a_3''. Such a stair-like structure will allow us to define EF formulas that play the roles of $counter_i$, $first_i$, $last_i$, $equal_i$, and $succ_i$ and their associated EF formulas in the previous proof. ◀

4.2 Lower Bounds for CTL

Data Complexity We know, for a fixed formula, model checking CTL, CTL+ and CTL* against HOPDSs is in k-ExpTime. Here, we show the lower bound.

▶ **Theorem 10.** *For a fixed formula, model checking CTL over a given order-k HOPDS without collapse is k-ExpTime-hard.*

Proof. (sketch) From Theorem 5 we take a language that is k-ExpTime-hard and fix an equivalent order-$(k-1)$ alternating HOPDA with a polynomially space-bounded work tape $\mathcal{P}$. The reduction is inspired by Bozzelli [4].

We use an order-k stack to navigate a computation tree of the HOPDA. To simulate the work tape, at each step, after an operation on the order-$(k-1)$ stack, a sequence of tape symbols are pushed on to the top order-1 stack. Then, the system can do a check branch to ensure the guessed tape is consistent with the previous, or continue simulating the execution. To continue, an order-k push saves the current state (for backtracking), the work tape is erased, the next rule is announced, and a $push_k$ remembers the rule. This process repeats. Consider the example order-3 stack below.

$$\left[\begin{array}{c}\left[\begin{array}{c}[tw_1]\\ [w_2]\\ \cdots\end{array}\right]\end{array}\right.\quad\left[\begin{array}{c}[r\ldots]\\ \cdots\end{array}\right]\quad\left[\begin{array}{c}[t'w_1']\\ [w_2']\\ \cdots\end{array}\right]\quad\left.\cdots\right]$$

This stack is at a configuration with the tape given by the word t and order-2 stack $[[w_1]\,[w_2]\ldots]$, which can backtrack to a configuration with tape t' and order-2 stack $[[w_1']\,[w_2']\ldots]$. The rule r connects the configurations. When an accepting configuration is seen, or the children of the current node have been fully explored, we backtrack using pop_k, and check untested universal branches. The automaton accepts when the (marked) initial stack is reached. That is, all paths have been explored, and found to be accepting.

The check branches have further branches for each of the polynomially many positions of the work tape. Each branch uses the control state to find the correct position, and then, using the control state, compares it with the corresponding positions in the previous work tape, which is recovered via pop_k operations.

The CTL formula $E((op \wedge AX(check \rightarrow AFgood))Ufin)$ asserts a path encoding an accepting tree exists, and checking branches all accept. The proposition *op* indicates the current path is simulating a tree, and *check* indicates a checking branch. Finally, *good* indicates that the check has been passed, and fin denotes the (successful) completion of the run. ◀

Expression Complexity The following theorem takes care of all cases.

▶ **Theorem 11.** *For a fixed order-k HOBPA without collapse, model checking CTL is k-ExpTime-hard.*

Proof. (sketch) The proof is in stages. First, we adapt Theorem 10, unfixing the formula to fix the HOPDS. A HOBPA is obtained using a more complex formula. There are two main assertions we move from the HOPDS to the formula. First, the check branch becomes a straight-line sequence of pops and the formula uses sequences of EX to compare positions. Secondly, the word position being read has to be guessed and added to the work tape information, then checked by the formula. Hence, we have the result for a fixed HOPDS.

To obtain a HOBPA the main difficulty is that control states were used to separate the check, backtrack and simulation phases of the model. Here we use the $(a, push_j, b)$ rules so that, when pushing, the automaton can read a character a, and mark it $\underline{a}$ in the next applied rule. Hence the system knows when it is moving up or down the stack, and when it has simulated a stack action. Also the automaton announces the intended phases. For example, the check branch announces "check", removes the work tape, announces "pop_k", pops, announces "check" again and removes the work tape. The formula can then use EX^j to look into the first tape, and $E(tapeU(pop_k \wedge EX(check \wedge EX^j\varphi)))$ to look j steps into the next tape, where *tape* indicates that a tape character is seen. ◀

4.3 Lower Bounds for CTL+

For data complexity, the CTL lower bound transfers to CTL+ and CTL*. For the expression complexity, the following theorem suffices.

▶ **Theorem 12.** *For a fixed order-k HOBPA without collapse, model checking CTL+ is* $(k+1)$*-ExpTime-hard.*

Proof. (sketch) First we adapt the proof of Theorem 10 to show, CTL* is $(k+1)$-ExpTime-hard. We then replace the CTL* formula with a CTL+ formula. Then we show how to fix the system, and restrict ourselves to HOBPA.

For a non-fixed formula and system, our CTL* proof adapts Bozzelli's order-1 proof [4]. Fix a language that is hard for $(k+1)$-ExpTime and an equivalent order-$(k-1)$ alternating HOPDA with an *exponentially* space-bounded work tape. The system proceeds as before, but guesses the length of the work tape and uses a word $bin_n(0)c_0bin_n(1)c_1 \cdots bin_n(2^n-1)c_{2^n-1}$ to represent it, where $bin_n(i)$ is the n-digit binary representation of i, and c_j are cell contents. The check phase has one branch to check the cell counters are sequential, and the others, instead of just popping down the stack, mark a position in each tape. The formula asserts, when markings are sensible, the tape contents are locally consistent. This can, in fact, be encoded in CTL+ by taking advantage of straight-line parts of the execution and adding extra markings. Obtaining a fixed HOBPA is similar to the CTL case, with some extra tricks. E.g., to ensure each marker is placed once and in the correct order. ◀

5 Linear Time

We consider the linear time logics. We first deal with the upper bound for linear time μ-calculus (μLTL) — and hence LTL — before considering the lower bounds in turn.

5.1 Upper Bounds for μLTL

Since the linear time μ-calculus (μLTL) does not translate polynomially into μ-calculus, we show the k-ExpTime upper bound of model checking μLTL against CPDS separately. Note that μLTL trivially subsumes all other linear time logics considered in this paper.

▶ **Theorem 13.** *Model checking μLTL against order-k CPDSs is in k-ExpTime for a non-fixed formula, and $(k-1)$-ExpTime for a fixed formula.*

Proof. (sketch) We can translate any μLTL formula φ into a Büchi automaton $\mathcal{B}$ at an exponential cost [29]. From a given CPDS $\mathcal{P}$ we construct a product CPDS $\mathcal{P}_B = \mathcal{P} \times \mathcal{B}$ which has a Büchi acceptance condition such that $\mathcal{P}_B$ accepts iff $\mathcal{P}$ does not satisfy φ.

An order-k Büchi CPDS is a CPDS parity game with two colours and only one player. Hence, non-emptiness can be reduced to determining the winner in a parity game, which takes k-ExpTime in the size of the CPDS [17]. Since the Büchi CPDS is exponential in the size of φ, this complexity is too high. The algorithm for an order-k parity game is by a reduction to an order-$(k-1)$ game of exponential size. Because the Büchi CPDS has one player, we can avoid the exponential blow up, constructing an order-$(k-1)$ game of polynomial size. This can be solved in $(k-1)$-ExpTime in the size of the Büchi CPDS, giving an algorithm in k-ExpTime for a non-fixed formula, and $(k-1)$-ExpTime for a fixed formula. ◀

5.2 Lower Bounds for LTL

We first give a matching lower bound for data complexity (fixed formula) of LTL, which already hold for its fragments LTL(F,X) and LTL(U). Since we have previously shown that order-k HOBPA can be analysed in P-time for fixed formulas, it remains to consider HOPDS.

▶ **Theorem 14.** *Model checking HOPDS without collapse against fixed LTL(F, X) and LTL(U) formulas is $(k-1)$-ExpTime-hard.*

Proof. The non-emptiness problem for HOPDS is $(k-1)$-ExpTime-complete [13]. This problem easily reduces to checking the fixed formula $G(\neg f)$, where f holds at all accepting states. Since this formula is both in LTL(F, X) and LTL(U), we are done. ◀

Next we study the expression complexity (fixed system). This is our main result of this section: already for a fixed order-k HOBPA, both LTL(F, X) and LTL(U) are k-ExpTime-hard.

▶ **Theorem 15.** *Model checking LTL(F, X) and LTL(U) against a fixed HOBPA without collapse is k-ExpTime-hard.*

Proof. (sketch) We take a k-ExpTime-hard language $\mathcal{L}$, and, by Theorem 5, its equivalent HOPDS with $s(n)$-bounded space work tape $\mathcal{P}$, for some polynomial $s(n)$. We shall construct a fixed HOBPA $\mathcal{P}'$ such that the language $\mathcal{L}$ is polynomial-time reducible to the LTL(F,X) model checking problem over $\mathcal{P}'$. We can similarly derive the desired lower bound for LTL(U) by "weakly" simulating the next operators with the until operator in the standard way.

We shall now give an intuition of the construction of $\mathcal{P}'$. Our HOBPA $\mathcal{P}'$ is an "over-approximation" of $\mathcal{P}$ in the sense that $\mathcal{P}'$ can do whatever actions $\mathcal{P}$ can do but also more. We will then use LTL(F,X) formulas to enforce correct simulations. This is of course due to the fact that $\mathcal{P}'$ lacks control states and work tape, and its definition should not depend on the input word to $\mathcal{P}$. We shall now elaborate more on how this can be implemented. Given a word $w = \alpha_1 \dots \alpha_n \in \Gamma^*$, we would like to determine if there is an accepting run of $\mathcal{P}$ on w, i.e., a sequence of configurations of the form $\langle p, \gamma, t, j\rangle$ starting with a starting configuration and ending with a final configuration. Here, p is a control state of $\mathcal{P}$, γ is a k-store, $t \in \Delta^{s(n)}$ is a tape content, and $1 \leq j \leq s(n)$ is the position of the tape head. We will represent each such configuration as the topmost symbols of a contiguous sequence of configurations of $\mathcal{P}'$. For example, suppose that the current configuration of $\mathcal{P}$ is $\langle p, \gamma, t, j\rangle$ where $top_1(\gamma) = a$. The HOBPA $\mathcal{P}'$ will start by having (p, a) as its topmost stack symbol. It will then keep modifing its topmost symbol to reflect the tape content t and the position j. This is done by guessing each individual tape cell content from left to right. At some point, $\mathcal{P}'$ will nondetermistically choose some rule of $\mathcal{P}$ to fire. We simulate this by first executing the stack operation and then guess some new state, which we put on top of the stack (this guess is needed because pop operations will destroy control state information). It will then continue by guessing next tape content in the same manner. This process can be repeated indefinitely, unless $\mathcal{P}'$ decides to go to a final (i.e. sink) state, in which $\mathcal{P}'$ will just loop forever. Given an input word $w = \alpha_1 \dots \alpha_n \in \Gamma^*$, we may force a correct simulation of $\mathcal{P}$ on w by $\mathcal{P}'$ using an LTL(F,X) formula. That is, we give a formula φ_w such that $w \in \mathcal{L}(\mathcal{P})$ iff $\mathcal{P}', c_0 \not\models \varphi_w$, where c_0 is an appropriate initial configuration of $\mathcal{P}'$ reflecting the initial state of $\mathcal{P}$. This can be done by first ensuring that each configuration of $\mathcal{P}$ in the simulation as a contiguous sequence of configurations of $\mathcal{P}'$ is valid. In particular, the guessed tape content (reflected by the topmost symbols in this sequence of configurations of $\mathcal{P}'$) must be of length $s(n)$ and has precisely one tape head, which can be easily expressed in LTL(F,X) using a single operator G and nestings of next operators of depth $s(n)$ (approximately). Recall that $s(n)$ is a polynomial function. Using the same technique, we also express that two representations of consecutive configurations of $\mathcal{P}$ in the simulation respect the transition relation of $\mathcal{P}$. Similarly, we enforce the initial configuration in the simulation and that some final configuration of $\mathcal{P}$ is reached. ◀

6 Future Work

There are several avenues of future work. E.g., we have no matching upper bound for the complexity of EF model checking. Walukiewicz has shown the problem to be PSpace-complete at order-1 [31]. However, his techniques do not easily extend to HOPDS owing to the subtleties of higher-order stacks. We may also study simpler logics such as LTL(F).

Acknowledgments. We thank Olivier Serre for interesting discussions and the anonymous referees for their helpful remarks. This work was partly supported by EPSRC (EP/F036361 and EP/E005039), and was done while the second author was a student at the School of Informatics, University of Edinburgh.

References

1 A. Bouajjani, J. Esparza, and O. Maler. Reachability analysis of pushdown automata: Application to model-checking. In *CONCUR*, p. 135–150, 1997.

2 A. Bouajjani, P. Habermehl, and R. Mayr. Automatic verification of recursive procedures with one integer parameter. *Theor. Comput. Sci.* 295:85–106 (2003)

3 A. Bouajjani and A. Meyer. Symbolic Reachability Analysis of Higher-Order Context-Free Processes. In *FSTTCS*, p. 135–147, 2004.

4 L. Bozzelli. Complexity results on branching-time pushdown model checking. *Theor. Comput. Sci.*, 379(1-2):286–297, 2007.

5 C. Broadbent and L. Ong. On global model checking trees generated by higher-order recursion schemes. In *FOSSACS*, p. 107–121, 2009.

6 O. Burkart, D. Caucal, F. Moller, and B. Steffen. Verification on infinite structures. In *Handbook of Process Algebra*, Elsevier, 1999.

7 T. Cachat. Higher order pushdown automata, the caucal hierarchy of graphs and parity games. In *ICALP*, p. 556–569, 2003.

8 T. Cachat and I. Walukiewicz. The complexity of games on higher order pushdown automata. *CoRR*, abs/0705.0262, 2007.

9 A. Carayol. Regular sets of higher-order pushdown stacks. In *MFCS*, p. 168–179, 2005.

10 A. Carayol and S. Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, p. 112–123, 2003.

11 D. Caucal. On infinite terms having a decidable monadic theory. In *Proc. MFCS*, p. 165–176, 2002.

12 E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Computer Science*, p. 995–1072, Elsevier, 1990.

13 J. Engelfriet. Iterated pushdown automata and complexity classes. In *STOC*, p. 365–373, 1983.

14 J. Esparza and J. Knoop. An automata-theoretic approach to interprocedural data-flow analysis. In *FoSSaCS*, p. 14–30, 1999.

15 E. M. Gurari. *An Introduction to the Theory of Computation*. W. H. Freeman & Co., New York, NY, USA, 1989.

16 M. Hague. *Global Model Checking Higher Order Pushdown Systems*. PhD thesis, Oxford University, 2009.

17 M. Hague, A. S. Murawski, C.-H. L. Ong, and O. Serre. Collapsible pushdown automata and recursion schemes. In *LICS*, p. 452–461, 2008.

18 T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, p. 205–222, 2002.

19 T. Knapik, D. Niwinski, P. Urzyczyn, I. Walukiewicz. Unsafe Grammars and Panic Automata. In *ICALP*, p. 1450-1461, 2005.

20 N. Kobayashi. Model-checking higher-order functions. In *PPDP*, p. 25–36, 2009.

21 N. Kobayashi. Types and higher-order recursion schemes for verification of higher-order programs. In *POPL*, p. 416–428, 2009.

22 N. Kobayashi and C.-H. L. Ong. Complexity of model checking recursion schemes for fragments of the modal mu-calculus. In *ICALP*, p. 223–234, 2009.

23 N. Kobayashi and C.-H. L. Ong. A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes. In *LICS*, p. 179–188, 2009.

24 A. N. Maslov. Multilevel stack automata. *Probl. Inf. Transm.*, 15:1170–1174, 1976.

25 R. Mayr. Strict lower bounds for model checking BPA. *Electr. Notes Theor. Comput. Sci.*, 18, 1998.

26 C.-H. L. Ong. On model-checking trees generated by higher-order recursion schemes. In *LICS*, p. 81–90, 2006.

27 A. Seth. An Alternative Construction in Symbolic Reachability Analysis of Second Order Pushdown Systems. Int. J. Found. Comput. Sci. 19(4): 983-998, 2008.

28 W. Thomas. Constructing Infinite Graphs with a Decidable MSO-Theory. In *MFCS*, p. 113–124, 2003.

29 M. Vardi. A temporal fixpoint calculus. In *POPL*, p. 250-259, 1988.

30 I. Walukiewicz. Pushdown processes: Games and model checking. In *CAV*, p. 62–74, 1996.

31 I. Walukiewicz. Model checking CTL properties of pushdown systems. In *FSTTCS*, p. 127–138, 2000.

A graph polynomial for independent sets of bipartite graphs*

Qi Ge and Daniel Štefankovič

Department of Computer Science
University of Rochester
Rochester, NY 14627, USA
{qge,stefanko}@cs.rochester.edu

Abstract

We introduce a new graph polynomial that encodes interesting properties of graphs, for example, the number of matchings, the number of perfect matchings, and, for bipartite graphs, the number of independent sets (#BIS).

We analyze the complexity of exact evaluation of the polynomial at rational points and show a dichotomy result—for most points exact evaluation is #P-hard (assuming the generalized Riemann hypothesis) and for the rest of the points exact evaluation is trivial.

We propose a natural Markov chain to approximately evaluate the polynomial for a range of parameters. We prove an upper bound on the mixing time of the Markov chain on trees. As a by-product we show that the "single bond flip" Markov chain for the random cluster model is rapidly mixing on constant tree-width graphs.

1998 ACM Subject Classification F.2.2, G.2.1, G.2.2, G.3

Keywords and phrases graph polynomials, #P-complete, independent sets, approximate counting problems, Markov chain Monte Carlo

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.240

1 Introduction

Graph polynomials are a well-developed area useful for analyzing properties of graphs (see, e. g., [7, 8] and [18]). Arguably the most intriguing graph polynomial is the Tutte polynomial [24, 25]. The partition function of the random cluster model from statistical mechanics provides a particularly simple definition: for a graph $G = (V, E)$ let

$$Z(G; q, \mu) = \sum_{S \subseteq E} q^{\kappa(S)} \mu^{|S|}, \tag{1}$$

where $\kappa(S)$ is the number of connected components of the graph (V, S). It is well-known that the Tutte polynomial is obtained from Z by a simple transformation, see, e. g., [28]. The Tutte polynomial includes many graph polynomials as special cases, e. g., the chromatic polynomial, the flow polynomial, and the Potts model (see, e. g., [28]).

Now we define our graph polynomial.

▶ **Definition 1.** The R_2-polynomial of a graph $G = (V, E)$ is

$$R_2(G; q, \mu) = \sum_{S \subseteq E} q^{\mathrm{rk}_2(S)} \mu^{|S|}, \tag{2}$$

* Research supported, in part, by NSF grant CCF-0910584.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 240–250

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

where $\mathrm{rk}_2(S)$ is the rank of the adjacency matrix of (V,S) over $\mathbb{F}_2$ (the field with 2 elements).

The most interesting fact about the R_2 polynomial is that for bipartite graphs it encodes the number of independent sets (see Theorem 4 below). We are not aware of any other graph polynomial that encodes the number of independent sets in a non-obvious manner. (The independence polynomial of graph G is $I(G;x)=\sum_k s_k x^k$, where s_k is the number of independent sets of G of size k; here, obviously $I(G,1)$ counts the number of independent sets of G.)

To illustrate a difference between the random cluster polynomial and the R_2-polynomial we provide a few small examples. Note that P_4 and claw graph (one vertex attached to 3 other vertices) have the same random cluster polynomial whereas C_3 and claw graph have the same R_2-polynomial.

	Random cluster polynomial	R_2 polynomial
claw graph	$(\mu+q)^3q$	$(\mu^3+3\mu^2+3\mu)q^2+1$
path P_4	$(\mu+q)^3q$	$(\mu^3+\mu^2)q^4+(2\mu^2+3\mu)q^2+1$
cycle C_3	$q^3+3\mu q^2+(\mu^3+3\mu^2)q$	$(\mu^3+3\mu^2+3\mu)q^2+1$

2 Our results

Now we look at how $R_2(G;q,\mu)$ encodes some properties of graphs.

▶ **Lemma 2.** *Substituting $q=\mu^{-1/2}$ into equation (2) we define*

$$P(G;\mu) := R_2(G;\mu^{-1/2},\mu) = \sum_{S\subseteq E(G)} \mu^{|S|-\mathrm{rk}_2(S)/2}.$$

Then $P(G;0)$ is the number of matchings in G.

Proof. Note that $\mathrm{rk}_2(S)\le 2|S|$ (since adding an edge to S changes two entries in the adjacency matrix and hence can change rank by at most two), and $\mathrm{rk}_2(S)<2|S|$ if S is not a matching (since rank of the adjacency matrix of a star is $2<2|S|$, and adding further edges preserves the strict inequality). ◀

▶ **Lemma 3.** *Let*

$$P(G;t,\mu) := t^{|V|}R_2(G;1/t,\mu) \quad \text{and} \quad P_2(G;\mu) := \mu^{-|V|/2}P(G;0,\mu).$$

Then $P_2(G;0)$ is the number of perfect matchings of G.

Proof. Note that only subsets with full rank adjacency matrix contribute to $P(G;0,\mu)$, and then only the minimal cardinality subsets with full rank adjacency matrix contribute to $P_2(G;0)$ (these subsets are exactly the perfect matchings). ◀

From now on we focus solely on bipartite graphs. For a *bipartite* graph $G=(U\cup W,E)$ we let

$$R_2'(G;\lambda,\mu) = \sum_{S\subseteq E} \lambda^{\mathrm{rk}_2(S)}\mu^{|S|}, \tag{3}$$

where $\mathrm{rk}_2(S)$ is the rank of the *bipartite* adjacency matrix of $(U\cup W,S)$. Note that

$$R_2(G;\lambda,\mu) = R_2'(G;\lambda^2,\mu), \tag{4}$$

since the adjacency matrix contains "two copies" of the bipartite adjacency matrix (one of them transposed). (The reason for definition (3) is that we prefer to operate with bipartite adjacency matrix for bipartite graphs.)

In Section 3 we prove that R'_2 counts the number of independent sets in bipartite graphs.

▶ **Theorem 4.** *Let $G = (U \cup W, E)$ be a bipartite graph. The number of independent sets of G is given by $2^{|U|+|W|-|E|}R'_2(G; 1/2, 1)$.*

Exact evaluation of the polynomial $R'_2(G; \lambda, \mu)$ is #P-hard at a variety of rational points (λ, μ) assuming the validity of the generalized Riemann hypothesis (GRH). The result is summarized in the following theorem.

▶ **Theorem 5.** *Exact evaluation of R'_2 at rational point (λ, μ) is*

- *polynomial-time computable when $\lambda \in \{0, 1\}$ or $\mu = 0$ or $(\lambda, \mu) = (1/2, -1)$;*
- *#P-hard when $\lambda \notin \{0, 1, 1/2\}$ and $\mu \neq 0$, assuming GRH;*
- *#P-hard when $\lambda = 1/2$ and $\mu \notin \{0, -1\}$.*

▶ Remark. For the non-bipartite case we have the following classification. Exact evaluation of R_2 at rational point (λ, μ) is polynomial-time computable when $\mu = 0$ or $\lambda \in \{-1, 0, 1\}$; the $\lambda = -1$ case follows from the fact that a skew-symmetric matrix with zero diagonal has even rank over any field (the zero diagonal condition is redundant for fields of characteristic $\neq 2$). For any other rational λ and μ we get #P-hardness of evaluating the R_2 polynomial from Theorem 5 and (4) (again assuming GRH). (Note that $(\lambda, \mu) \mapsto (\lambda^2, \mu)$ never maps to the easy case $(1/2, -1)$, since λ is rational. It would be nice to have hardness classification of evaluating R_2 and R'_2 for, say, algebraic λ and μ.)

Because of the hardness of exact evaluation of R'_2, we turn to approximate evaluation of $R'_2(G; \lambda, \mu)$.

We now define the sampling problem associated with R'_2.

Rank Weighed Subgraphs with $\lambda, \mu \geq 0$, (RWS(λ, μ))
Instance: a bipartite graph $G = (U \cup W, E)$,
Output: $S \subseteq E$ with probability of $S \propto \lambda^{\mathrm{rk}_2(S)}\mu^{|S|}$.

The "single bond flip" chain is a natural approach to sampling from RWS(λ, μ).

▶ **Definition 6.** *Single bond flip* chain is defined as follows: pick an edge $e \in E$ at random and let $S = X_t \oplus \{e\}$. Set $X_{t+1} = S$ with probability

$$(1/2)\min\{1, \lambda^{\mathrm{rk}_2(S)-\mathrm{rk}_2(X_t)}\mu^{|S|-|X_t|}\}$$

and $X_{t+1} = X_t$ with the remaining probability.

In each step of the single bond flip chain, we have to compute the rank of a matrix over $\mathbb{F}_2$ (corresponding to S) which differs from the current matrix (corresponding to X_t) in a single entry. One can use dynamic matrix rank problem algorithms to perform this computation in $O(n^{1.575})$ arithmetic operations per step [9].

Instead of flipping one edge in a step, we can have another Markov chain which flips a random subset of edges adjacent to a single vertex. It seems likely that the new chain can generate good random samples faster than the single bond flip chain—a step of the new chain can be performed in $O(n^2)$ arithmetic operations (using "rank one update" for the dynamic matrix rank problem [9]).

We consider the following question.

▶ Question 1. For which classes of bipartite graphs does the single bond flip chain mix?

In Section 5 we prove that for fixed $\lambda, \mu > 0$ the single bond flip chain mixes, in time polynomial in the number of vertices, for trees. The next theorem is motivated by Question 1, (R'_2 polynomial can be evaluated in polynomial time on trees).

▶ **Theorem 7.** *For every fixed $\lambda, \mu > 0$, the mixing time $\tau(\varepsilon)$ of the single bond flip chain for a tree on n vertices is*

$$\tau(\varepsilon) = O\left(n^{3+|\log_2 \lambda|}(|\log \lambda| + |\log \mu| + \log(1/\varepsilon))\right).$$

▶ Remark. Goldberg and Jerrum [12] recently showed that there exist bipartite graphs for which the single bond flip chain needs exponential time to mix for $\lambda = 1/2$ and $\mu = 1$ (which is the most interesting setting of λ and μ). Question 1 is still relevant—there may exist interesting classes of graphs for which the chain mixes.

As a by-product of our techniques, we show that single bond flip Markov chain for the random cluster model is rapidly mixing if $q, \mu > 0$ and G has constant tree-width (the condition $q, \mu > 0$ is equivalent to $x, y > 1$ for the Tutte polynomial $T(G; x, y)$).

Due to page limitation, we omit most of the proofs. Refer to [10] for a full version.

3 Independent sets in bipartite graphs

The problem of counting independent sets (#IS) in a graph is of interest in both computer science and statistical physics (independent sets are a special case of the so-called hard-core model, see, e. g., [1]). Exact computation of #IS is #P-complete even for 3-regular planar bipartite graphs [26, 29]. Fully polynomial randomized approximation scheme (FPRAS) is known for graphs with maximum degree $\Delta \leq 5$, [17, 6, 27]. Unless NP=RP, an FPRAS does not exist for graphs with $\Delta \geq 6$, [3, 21] .

Now we focus on the problem of counting independent sets in bipartite graphs (#BIS). While for exact counting the complexity of #BIS and #IS is the same, the situation looks very different for approximate counting, for example, no inapproximability result is known for #BIS. Dyer et al. [4] show that #BIS is complete w.r.t. approximation-preserving reductions (AP-reductions) in a sub-class of #P. Many problems were shown to be equivalent (w.r.t. AP-reductions) to #BIS, for example, #Downsets, #1p1nSat [4], computing the partition function of a ferromagnetic Ising model with local fields [11], and counting the number of satisfying assignments of a class of Boolean CSP instances [5]. A pertinent negative result for #BIS is that Glauber dynamics (or more generally, any chain whose states are independent sets and that flips at most $0.35n$ vertices in one step) cannot be used to efficiently sample random independent sets in a random 6-regular bipartite graphs on $n + n$ vertices [3].

The rest of this section is devoted to proving Theorem 4. It will be convenient to work with matrices instead of graphs. For two zero-one matrices A, B we say $B \leq A$ if B corresponds to a subgraph of A, formally

▶ **Definition 8.** Let A, B be zero-one $n_1 \times n_2$ matrices. We say $B \leq A$ if $A_{ij} = 0$ implies $B_{ij} = 0$, for all $i \in [n_1]$ and $j \in [n_2]$. Let $\mathcal{C}_A$ be the set of zero-one $n_1 \times n_2$ matrices B such that $B \leq A$.

Let $\#_1(A)$ denote the number of ones in A (that is, the number of edges in the corresponding graph). The RWS problem rephrased for matrices is:

Rank Weighed Matrices with $\lambda, \mu \geq 0$ (RWM(λ, μ))

Instance: an $n_1 \times n_2$ matrix A.
Output: $B \in \mathcal{C}_A$ with probability of $B \propto \lambda^{\mathrm{rk}_2(B)}\mu^{\#_1(B)}$.

The problem of sampling independent sets in bipartite graphs is:

Bipartite Independent Sets (BIS)
Instance: a bipartite graph $G = (U \cup W, E)$.
Output: a uniformly random independent set of G.

Before we show a connection between BIS and RWM$(1/2, 1)$ we remark that to sample bipartite independent sets it is enough to sample a subset of one side, say U, from the correct (marginal) distribution. We now describe this distribution in a setting which will be advantageous for the proof of Theorem 4.

We will represent an independent set by a pair of (indicator) vectors u, v (where $u \in \mathbb{F}_2^{n_1}$ and $v \in \mathbb{F}_2^{n_2}$).

▶ **Definition 9.** We say that two vectors $\alpha, \beta \in \mathbb{F}_2^n$ *share a one* if there exists $i \in [n]$ such that $\alpha_i = \beta_i = 1$.

We will use the following simple fact.

▶ Observation 1. Let $\alpha, \beta \in \mathbb{F}_2^n$. Let d be the number of ones in β. If α, β share a one then there are 2^{d-1} vectors $\beta' \leq \beta$ such that $\alpha^{\mathrm{T}}\beta' \equiv 0 \mod 2$. If α, β do not share a one then there are 2^d vectors $\beta' \leq \beta$ such that $\alpha^{\mathrm{T}}\beta' \equiv 0 \mod 2$.

Let $u \in \mathbb{F}_2^{n_1}$ be a vector. We would like to count the number of $v \in \mathbb{F}_2^{n_2}$ such that u, v is an independent set. Note that u, v is an independent set iff $v_j = 0$ for every $j \in [n_2]$ such that u and j-th column of A share a one. Let k be the number of columns of A that do not share a one with u. Then we have

$$u \in \mathbb{F}_2^{n_1} \text{ occurs in } 2^k \text{ independent sets.} \tag{5}$$

Thus to sample independent sets in a bipartite graph G with $n_1 \times n_2$ bipartite adjacency matrix A it is enough to sample $u \in \mathbb{F}_2^{n_1}$ with the probability of u proportional to 2^k, where k is the number of columns of A that do not share a one with u. We will call this distribution on u the marginal BIS distribution.

The following lemma shows a tight connection between BIS and RWM$(1/2, 1)$—given a sample from one distribution it is trivial to obtain a sample from the other one.

▶ **Lemma 10.** *Let G be a bipartite graph with bipartite adjacency matrix A.*

- *Let u, v be a uniformly random independent set of G. Let B be a uniformly random matrix from the following set $\{D \in \mathcal{C}_A \mid u^{\mathrm{T}}D \equiv 0 \mod 2\}$. Then B is from the* RWM$(1/2, 1)$*-distribution.*
- *Let $B \in \mathcal{C}_A$ be a random matrix from the* RWM$(1/2, 1)$*-distribution. Let $u \in \mathbb{F}_2^{n_1}$ be a uniformly random vector from the left null space of B (that is, $\{\beta \in \mathbb{F}_2^{n_1} \mid \beta^{\mathrm{T}}B \equiv 0 \mod 2\}$). Then u is from the marginal* BIS *distribution.*

Proof. Let Q be the set of u, B pairs such that $u^{\mathrm{T}}B \equiv 0 \mod 2$ and $B \leq A$. Let ψ be the uniform distribution on Q. Note that ψ marginalized over u yields the RWM$(1/2, 1)$-distribution on $B \leq A$, here we are using the fact that a d-dimensional space (in this case the left null space of B) over $\mathbb{F}_2$ has 2^d elements. Formally,

$$P(B) = \sum_{u : u^{\mathrm{T}}B \equiv 0 \mod 2} \frac{1}{|Q|} = \frac{2^{n_1 - \mathrm{rk}_2(B)}}{|Q|} = \frac{2^{-\mathrm{rk}_2(B)}}{R'_2(G; 1/2, 1)}. \tag{6}$$

Next we show that ψ marginalized over B yields the marginal BIS distribution. We compute the number of $B \leq A$ such that $u^{\mathrm{T}}B \equiv 0 \mod 2$. Let us use the same k as in (5), that is, k is the number of columns of A that do not share a one with u.

Note that the columns of B can be chosen independently and only if the column and u share a one is the number of choices (for that column) halved. Let $\#_1(A)$ be the number of ones in A. Thus

$$\text{there are } 2^{\#_1(A)-(n_2-k)} \text{ choices of } B \leq A \text{ such that } u^{\mathrm{T}}B \equiv 0 \mod 2. \tag{7}$$

Note that for fixed u the counts in (5) and (7) differ by a factor of $2^{\#_1(A)-n_2}$ (which is independent of u). Thus ψ marginalized over B yields the marginal BIS distribution on u. Formally

$$P(u) = \frac{2^{\#_1(A)-(n_2-k)}}{|Q|} = \frac{2^k}{\#\mathrm{BIS}(G)}. \tag{8}$$

Note that this proves both claims of the lemma since in both cases the u, B pair is from ψ (by first sampling from a marginal and then sampling the remaining variable) and the conclusion in both claims is a statement about marginal (of the remaining variable). ◀

Theorem 4 now follows from the proof of Lemma 10.

Proof of Theorem 4. Let Q be the set from the proof of Lemma 10. From (6) we obtain

$$|Q| = R'_2(G; 1/2, 1)2^{n_1}. \tag{9}$$

From (8) we have that the number of independent sets of G is given by

$$\#\mathrm{BIS}(G) = \frac{|Q|}{2^{\#_1(A)-n_2}}. \tag{10}$$

Combining (9) and (10) we obtain the theorem. ◀

We do not know a good combinatorial interpretation for the mod-2 rank of B for general graphs. For forests (which are, of course, always bipartite) we have the following characterization.

▶ **Lemma 11.** *Let $G = (V, E) = (U \cup W, E)$ be a forest with bipartite adjacency matrix A. Then $\mathrm{rk}_2(A)$ is the size of maximum matching in G.*

Proof. Let $a \in V$ be a leaf of G and let $e = \{a, b\} \in E$ be the edge adjacent to a. Note that b is matched in every maximum matching M (otherwise one could add e to M). Thus removing b and all adjacent edges decreases the size of maximum matching by 1.

Now we argue that removing b (and all adjacent edges) also decreases rank (over $\mathbb{F}_2$) by 1. W.l.o.g. assume that b corresponds to the first row and a corresponds to the first column. Removing b (and all adjacent edges) corresponds to removing the first row of A. Note that this decreases rank by at most 1 and it does decrease it by 1, since the only non-zero entry in the first column is in the first row. ◀

4 The linear-width of a graph

For the proof of Theorem 7 we will use the linear-width of a graph, a concept which was first defined by Thomas [23]. In this section we prove a bound on linear-width in terms of tree-width.

The *linear-width* of a graph $G = (V, E)$ is the smallest integer ℓ such that the edges of G can be arranged in a linear order $e_1, \ldots, e_m$ in such a way that, for every $i \in [m]$, there are at most ℓ vertices that have an adjacent edge in $\{e_1, \ldots, e_{i-1}\}$ and an adjacent edge in $\{e_i, \ldots, e_m\}$. It is known that computing the linear-width of a graph is NP-complete [22]. For paths and cycles the linear-width is easy to compute.

▶ **Example 12.** The linear-width of a path is 1. The linear-width of a cycle is 2.

Let $e_1, \ldots, e_m$ be a permutation of the edges of $G = (V, E)$. We say that a vertex $v \in V$ is *dangerous* w.r.t. $i \in [m]$, if there exist two edges e_j, e_k adjacent to v such that $j < i \leq k$. Let D_i be the set of vertices which are dangerous w.r.t. i. Note that the linear-width of G is the minimum value of $\max_i |D_i|$ optimized over all permutations of the edges.

Now we give an upper bound on the linear-width for trees.

▶ **Lemma 13.** *Let $T = (V, E)$ be a tree on n vertices. The linear-width of T is at most $\lfloor \log_2 n \rfloor$.*

For general graphs we will show a generalization of Lemma 13: a bound on the linear-width of G in terms of the tree-width of G. We now define tree-width (see, e. g., [16] for a nice treatment).

Given a graph $G = (V, E)$, a *tree decomposition* of G is a pair $(T, \{U_h\}_{h \in V_T})$ where $T = (V_T, E_T)$ is a tree and $U_h \subseteq V$ satisfy: (i) each edge of G is in at least one subgraph induced by U_h; and (ii) for any three vertices t_1, t_2, t_3 of T such that t_2 is in the path between t_1 and t_3 in T we have $U_{t_1} \cap U_{t_3} \subseteq U_{t_2}$. The *width* of a decomposition is $\max_{h \in V_T} |U_h| - 1$. The *tree-width* of G (denoted $\mathrm{tw}(G)$) is the minimum width optimized over all tree decompositions.

▶ **Lemma 14.** *Let $G = (V, E)$ be a graph. Then*

$$\text{linear-width}(G) \leq (\mathrm{tw}(G) + 1)(\lfloor \log_2 n \rfloor + 1).$$

5 Analysis of the single bond flip chain for trees

Given a tree $G = (V, E)$, let Ω be the set of $2^{|E|}$ subsets of E. By Lemma 11, for every $H \subseteq E$, we know that $\mathrm{rk}_2(H)$ is the size of maximum matching of the subgraph (V, H). Let $w(H)$ be the size of maximum matching in a graph (V, H). Let P be the transition matrix of the single bond flip Markov chain $\mathcal{M}$ from definition 6. It's easy to see that $\mathcal{M}$ is ergodic with unique stationary distribution π such that $\pi(H) \propto \lambda^{w(H)} \mu^{|E|}$.

The goal of this section is to prove Theorem 7.

5.1 The canonical paths

We will bound the mixing time of our chain $\mathcal{M}$ using the canonical paths method, introduced in [2, 20, 15]. Now we go over the basic definitions for Markov chains, see, e. g., [14] for a comprehensive background.

▶ **Definition 15.** The *total variation distance* of two probability distribution ν and ν' on Ω is

$$\|\nu - \nu'\|_{TV} = \frac{1}{2}\sum_{H\in\Omega}|\nu(H) - \nu'(H)| = \max_{\mathcal{S}\subseteq\Omega}|\nu(\mathcal{S}) - \nu'(\mathcal{S})|.$$

▶ **Definition 16.** The *mixing time* from initial state H, $\tau_H(\varepsilon)$, is defined as

$$\tau_H(\varepsilon) = \min\{t : \|P^t(H,\cdot) - \pi\|_{TV} \le \varepsilon\},$$

and the *mixing time* $\tau(\varepsilon)$ of the chain is defined as $\tau(\varepsilon) = \max_{H\in\Omega}\{\tau_H(\varepsilon)\}$.

Let $\sigma = e_1, \ldots, e_m$ be an ordering of the edges of $G = (V, E)$ (we will usually use the orderings supplied by Lemma 13 or Lemma 14). Given any pair $I, F \in \Omega$, let $I \oplus F$ be the symmetric difference of I and F (that is, the set of edges which are in either I or F but not in both). We define a canonical path $\gamma_{I,F}$ between I and F as follows. Let $e_{i_1}, \ldots, e_{i_k}$ be the edges from $I \oplus F$ ordered according to σ (that is, $i_1 < i_2 < \cdots < i_k$). Let

$$\gamma_{I,F} = (H_0, H_1, \ldots, H_k), \tag{11}$$

where $H_0 = I$, $H_k = F$ and $H_j = H_{j-1} \oplus \{e_{i_j}\}$.

▶ **Lemma 17.** *Let $G = (V, E)$ be a graph. Let $\sigma = e_1, \ldots, e_m$ be an ordering on E with linear-width ℓ. Let I, F be subsets of E and let H be on the canonical path* (11) *(that is, $H = H_j$ for some $j \in \{0, \ldots, k\}$). Then*

$$|w(I) + w(F) - w(H) - w(C)| \le \ell,$$

where $C = I \oplus F \oplus H$, (and $w(S)$ is the size of the maximum matching in (V, S)).

Proof. Let $Q = \{e_1, \ldots, e_{i_j}\}$. Note that $H = (F \cap Q) \cup (I \cap Q^c)$, where Q^c is the complement of Q (that is, $E \setminus Q$). Similarly, $C = (I \cap Q) \cup (F \cap Q^c)$.

Let D be the set of dangerous vertices w.r.t. e_{i_j+1}. Let M_I and M_F be maximum matchings of I and F, respectively. Let

$$M_H = (M_F \cap Q) \cup (M_I \cap Q^c) \quad \text{and} \quad M_C = (M_I \cap Q) \cup (M_F \cap Q^c).$$

Note that all vertices of M_H with degree ≥ 2 are in D (a vertex which is not D has all adjacent edges (in G) from Q or from Q^c and hence the adjacent edges (in M_H) agree with M_I or M_F). The same is true for M_C. Moreover if a vertex $v \in D$ has degree 2 in M_H then it has degree 0 in M_C. Thus by removing $\le |D|$ edges from M_H and M_C we can turn both of them into matchings. Thus

$$w(H) + w(C) \ge w(I) + w(F) - |D| \ge w(I) + w(F) - \ell. \tag{12}$$

Note that a canonical path from $I' := H$ to $F' := C$ passes through $H' := I$ (with $C' := I' \oplus F' \oplus H' = F$). Thus

$$w(I) + w(F) = w(H') + w(C') \ge w(I') + w(F') - \ell = w(H) + w(C) - \ell. \tag{13}$$

Combining (12) and (13) we get the lemma. ◀

5.2 The congestion of $\mathcal{M}$

Now we analyze the congestion of the collection $\Gamma = \{\gamma_{I,F} \mid I, F \in \Omega\}$ where $\gamma_{I,F}$ are canonical paths defined in (11). For each transition (H, H') such that $P(H, H') > 0$, let $cp(H, H')$ be the set of pairs (I, F) such that $(H, H') \in \gamma_{I,F}$. The congestion of Γ on (H, H') is (see, e. g., [14])

$$\varrho_{(H,H')} = \frac{1}{P(H,H')} \sum_{I,F:(H,H')\in\gamma_{I,F}} \frac{\pi(I)\pi(F)}{\pi(H)} |\gamma_{I,F}|, \tag{14}$$

where $|\gamma_{I,F}|$ is the length of $\gamma_{I,F}$. The congestion of Γ is defined as

$$\varrho := \max_{\substack{(H,H'):\\ P(H,H')>0}} \varrho_{(H,H')}.$$

We will use the following connection between the congestion and the mixing time.

▶ **Theorem 18** ([2, 20]). *$\tau_H(\varepsilon) \le \varrho(\log(1/\pi(H)) + \log(1/\varepsilon))$ for each starting state $H \in \Omega$.*

At the end of this section we prove the following bound on the congestion of Γ.

▶ **Lemma 19.** *Let $G = (V, E)$ be a graph. Let $\sigma = e_1, \ldots, e_m$ be an ordering on E with linear-width ℓ. For every (H, H') such that $P(H, H') > 0$, and for every $\lambda, \mu > 0$ we have $\varrho_{(H,H')} \le 2|E|^2 \bar{\lambda}^\ell$, where $\bar{\lambda} = \max\{\lambda, 1/\lambda\}$.*

We can now prove Theorem 7.

Proof of Theorem 7. Since $G = (V, E)$ is a tree, by Lemma 13, we have $\ell \le \lfloor \log_2 n \rfloor$, by Lemma 19, we have

$$\varrho \le 2|E|^2 \bar{\lambda}^\ell \le 2|E|^2 n^{|\log_2 \lambda|} \le 2n^{2+|\log_2 \lambda|}.$$

Theorem 7 now follows from Theorem 18. ◀

Now we bound the congestion of our canonical paths.

Proof of Lemma 19. We will bound $\varrho_{(H,H')}$ for every (H, H') such that $P(H, H') > 0$. Let $\hat{H} = H$ if $\pi(H) \le \pi(H')$ and $\hat{H} = H'$ otherwise. Note that

$$\frac{\pi(\hat{H})}{2|E|} = \pi(H)P(H, H') = \pi(H')P(H', H), \tag{15}$$

since $\mathcal{M}$ is reversible. We define a mapping $f : cp(H, H') \to \Omega$ such that $f(I, F) = I \oplus F \oplus \hat{H}$ for every pair $(I, F) \in cp(H, H')$.

First, note that f is an injection. Given $J \in \Omega$ we can determine the unique I, F such that $f(I, F) = J$, by first computing $J \oplus \hat{H}$, and the using the ordering σ on the edges of G to recover I and F.

Note that

$$|I| + |F| = |\hat{H}| + |f(I, F)|, \tag{16}$$

and

$$|w(I) + w(F) - w(\hat{H}) - w(f(I, F))| \le \ell, \tag{17}$$

where (17) follows from Lemma 17.

Let $L = \sum_J \lambda^{w(J)}\mu^{|J|}$. We have the following upper bound on $\varrho_{(H,H')}$. By (14) and (15), we have

$$\begin{aligned}
\varrho_{(H,H')} &= 2|E| \sum_{(I,F)\in cp(H,H')} \frac{\pi(I)\pi(F)}{\pi(\hat{H})}|\gamma_{I,F}| \\
&= 2|E|^2 \sum_{(I,F)\in cp(H,H')} \frac{\lambda^{w(I)+w(F)-w(\hat{H})}\mu^{|I|+|F|-|\hat{H}|}}{L} \\
&\le 2|E|^2\bar{\lambda}^{\ell} \sum_{(I,F)\in cp(H,H')} \frac{\lambda^{w(f(I,F))}\mu^{|f(I,F)|}}{L} \qquad (18)\\
&\le 2|E|^2\bar{\lambda}^{\ell}, \qquad (19)
\end{aligned}$$

where (18) follows from (16) and (17), and (19) follows from the fact that f is an injection from $cp(H, H')$ to Ω. ◀

6 Conclusions

We conclude with an observation that a generalization of RWM(λ, μ) does not have an FPRAS (unless NP=RP) and a few questions.

Let A be an $m \times n$ matrix whose entries are zeros, ones, and indeterminates, where each indeterminate occurs once. A *completion* of A is a substitution of $0, 1$ to all the indeterminates in A. We denote $\mathcal{C}_A$ to be the set of all completions of A. Let $\mathrm{rk}_2(B)$ be the rank of B over $\mathbb{F}_2$. Can we sample B from $\mathcal{C}_A$ with the probability of B proportional to $\lambda^{\mathrm{rk}_2(B)}$? Note that this problem is a generalization of the RWM$(\lambda, 1)$ problem. It turns out that finding the minimum rank completion of a matrix is NP-hard (Proposition 2.1, [19]) and hence a sampler is unlikely (unless NP=RP), since for $\lambda = 2^{-n^2}$ a random completion will be the minimum rank completion (with constant probability). The sampling problem could be easy for sufficiently large λ (the problem of finding maximum rank completion is in P, see, e. g., Section 4.1 of [13]).

▶ Question 2. What other interesting properties are encoded by the polynomial?

▶ Question 3. Can one sample maximum rank completions of a matrix?

▶ Question 4. Is the exact evaluation of the polynomial easy for bounded tree-width graphs?

References

1 Rodney J. Baxter. *Exactly solved models in statistical mechanics*. Academic Press Inc. [Harcourt Brace Jovanovich Publishers], London, 1989. Reprint of the 1982 original.

2 Persi Diaconis and Daniel Stroock. Geometric bounds for eigenvalues of Markov chains. *Ann. Appl. Probab.*, 1(1):36–61, 1991.

3 Martin Dyer, Alan Frieze, and Mark Jerrum. On counting independent sets in sparse graphs. *SIAM J. Comput.*, 31(5):1527–1541 (electronic), 2002.

4 Martin Dyer, Leslie Ann Goldberg, Catherine Greenhill, and Mark Jerrum. The relative complexity of approximate counting problems. *Algorithmica*, 38(3):471–500, 2004. Approximation algorithms.

5 Martin Dyer, Leslie Ann Goldberg, and Mark Jerrum. An approximation trichotomy for boolean #CSP. *Journal of Computer and System Sciences*, In Press, Corrected Proof, 2009.

6 Martin Dyer and Catherine Greenhill. On Markov chains for independent sets. *J. Algorithms*, 35(1):17–49, 2000.

7 Joanna Ellis-Monaghan and Criel Merino. Graph polynomials and their applications I: the Tutte polynomial. *arXiv*, 0803.3079, Jun 2008.

8 Joanna Ellis-Monaghan and Criel Merino. Graph polynomials and their applications II: interrelations and interpretations. *arXiv*, 0806.4699, Jun 2008.

9 Gudmund Skovbjerg Frandsen and Peter Frands Frandsen. Dynamic matrix rank. In *Automata, languages and programming. Part I*, volume 4051 of *Lecture Notes in Comput. Sci.*, pages 395–406. Springer, Berlin, 2006.

10 Qi Ge and Daniel Štefankovič. A graph polynomial for independent sets of bipartite graphs. *arXiv*, 0911.4732, Jan 2010.

11 Leslie Ann Goldberg and Mark Jerrum. The complexity of ferromagnetic Ising with local fields. *Combin. Probab. Comput.*, 16(1):43–61, 2007.

12 Leslie Ann Goldberg and Mark Jerrum. Personal communication, 2010.

13 Nicholas J. A. Harvey, David R. Karger, and Kazuo Murota. Deterministic network coding by matrix completion. In *Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 489–498 (electronic), New York, 2005. ACM.

14 Mark Jerrum. *Counting, sampling and integrating: algorithms and complexity*. Lectures in Mathematics ETH Zürich. Birkhäuser Verlag, Basel, 2003.

15 Mark Jerrum and Alistair Sinclair. Approximating the permanent. *SIAM J. Comput.*, 18(6):1149–1178, 1989.

16 Jon Kleinberg and Eva Tardos. *Algorithm Design*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

17 Michael Luby and Eric Vigoda. Fast convergence of the Glauber dynamics for sampling independent sets. *Random Structures Algorithms*, 15(3-4):229–241, 1999. Statistical physics methods in discrete probability, combinatorics, and theoretical computer science (Princeton, NJ, 1997).

18 Johann A. Makowsky. From a zoo to a zoology: towards a general theory of graph polynomials. *Theory Comput. Syst.*, 43(3-4):542–562, 2008.

19 René Peeters. Orthogonal representations over finite fields and the chromatic number of graphs. *Combinatorica*, 16(3):417–431, 1996.

20 Alistair Sinclair. Improved bounds for mixing rates of Markov chains and multicommodity flow. *Combin. Probab. Comput.*, 1(4):351–370, 1992.

21 Allan Sly. Computational transition at the uniqueness threshold. *arXiv*, 1005.5584, May 2010.

22 Dimitrios M. Thilikos. Algorithms and obstructions for linear-width and related search parameters. *Discrete Appl. Math.*, 105(1-3):239–271, 2000.

23 Robin Thomas. Tree-decompositions of graphs (lecture notes), 1996.

24 William T. Tutte. A ring in graph theory. *Proc. Cambridge Philos. Soc.*, 43:26–40, 1947.

25 William T. Tutte. A contribution to the theory of chromatic polynomials. *Canadian J. Math.*, 6:80–91, 1954.

26 Salil P. Vadhan. The complexity of counting in sparse, regular, and planar graphs. *SIAM J. Comput.*, 31(2):398–427 (electronic), 2001.

27 Dror Weitz. Counting independent sets up to the tree threshold. In *STOC'06: Proceedings of the 38th Annual ACM Symposium on Theory of Computing*, pages 140–149. ACM, New York, 2006.

28 Dominic J. A. Welsh. *Complexity: knots, colourings and counting*, volume 186 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, Cambridge, 1993.

29 Mingji Xia, Peng Zhang, and Wenbo Zhao. Computational complexity of counting problems on 3-regular planar graphs. *Theoret. Comput. Sci.*, 384(1):111–125, 2007.

Finding Independent Sets in Unions of Perfect Graphs

Venkatesan T. Chakaravarthy, Vinayaka Pandit, Sambuddha Roy, and Yogish Sabharwal

IBM Research - India, New Delhi
{vechakra, pvinayak, sambuddha, ysabharwal}@in.ibm.com

Abstract

The maximum independent set problem (MaxIS) on general graphs is known to be NP-hard to approximate within a factor of $n^{1-\epsilon}$, for any $\epsilon > 0$. However, there are many "easy" classes of graphs on which the problem can be solved in polynomial time. In this context, an interesting question is that of computing the maximum independent set in a graph that can be expressed as the union of a small number of graphs from an easy class. The (MaxIS) problem has been studied on unions of interval graphs and chordal graphs. We study the (MaxIS) problem on unions of perfect graphs (which generalize the above two classes). We present an $O(\sqrt{n})$-approximation algorithm when the input graph is the union of two perfect graphs. We also show that the (MaxIS) problem on unions of two comparability graphs (a subclass of perfect graphs) cannot be approximated within any constant factor.

1998 ACM Subject Classification F.2.2 [Analysis of Algorithms]

Keywords and phrases Approximation Algorithms; Comparability Graphs; Hardness of approximation

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.251

1 Introduction

It is well known that the classical maximum independent set problem (MaxIS) on general graphs is computationally hard, even to approximate. Zuckerman [13] showed that for any $\epsilon > 0$, it is NP-hard to approximate the MaxIS problem within a factor of $n^{1-\epsilon}$, where n is the number of vertices in the input graph. However, there are *easy* families of graphs for which the MaxIS problem can be solved optimally in polynomial time, for example interval graphs. In this context, it is interesting to consider the MaxIS problem on graphs which are unions of a small number graphs from an easy family. In this paper, we study this problem on perfect graphs and its subclass comparability graphs.

Formally, the input consists of a sequence of graphs $G_1 = (V, E_1), G_2 = (V, E_2), \ldots, G_t = (V, E_t)$ defined over the same vertex set V. A subset of vertices $X \subseteq V$ is called a *common independent set* (CIS), if X is an independent set in each graph G_i. Alternatively, a CIS is an independent set in the *union* graph given by $\widehat{G} = (V, \widehat{E})$, where $\widehat{E} = \cup_{i=1}^{t} E_i$. The goal is to find the maximum cardinality CIS. We call this the *maximum common independent set problem* (MaxCIS). For a fixed constant k, the k-MaxCIS is the special case where the number of input graphs is $t = k$. We consider restricted versions of the MaxCIS problem wherein all the input graphs belong to a particular class (or family) of graphs $\mathcal{C}$. This paper deals with the MaxCIS problem on easy classes $\mathcal{C}$, such as interval, chordal, comparability and perfect graphs. We shall also consider the weighted versions of MaxIS and MaxCIS problem, wherein the input includes a function $p : V \to \mathbf{R}$ that assigns a profit (or weight)

$p(u)$ to each vertex u and the goal is to find the maximum profit independent set and CIS, respectively.

Motivated by applications in bioinformatics, scheduling and computational geometry, the MaxCIS problem on interval graphs has been well-studied (see [1]). Bar-Yehuda et al. [1] presented a $2t$-approximation algorithm for the weighted MaxIS problem on the class of t-interval graphs, which includes unions of t intervals graphs. Hermelin and Rawitz [9] generalized their result by presenting a $2t$-approximation algorithm for class of t-subtrees, which includes both $2t$-interval graphs and chordal graphs. Regarding unions of comparability and perfect graphs, prior work deals with certain combinatorial aspects. It is well known that any perfect graph G on n vertices has either an independent set of size $\sqrt{n}$ or a clique of size $\sqrt{n}$. Motivated by applications in computational geometry, Dumitrescu and Tóth [5] studied the same issue on unions of comparability graphs and perfect graphs, from a combinatorial perspective. They showed that if G is the union of two perfect graphs, then G has an independent set of size $n^{1/3}$ or a clique of size $n^{1/3}$. They also provided counter-examples to show that the above bound cannot be improved beyond $n^{0.42}$ (see Theorem 5).

The above discussion shows that approximation algorithms are known for the weighted MaxCIS problem on interval graphs and chordal graphs; both these classes are easy classes for which the weighted MaxIS problem can be solved optimally in polynomial time [7, 6]. The goal of this paper study the MaxCIS problem on comparability graphs and perfect graphs, two other important easy classes for which the MaxIS problem can be solved optimally in polynomial time [12, 8]. Perfect graphs generalize the other classes mentioned above and can be thought of as the pinnacle among easy classes.

Our main result presents an $O(\sqrt{n})$-approximation algorithm for the weighted 2-MaxCIS problem on perfect graphs. The algorithm is obtained by considering a suitable LP formulation. The LP is of exponential size. We solve it using a separation oracle and find the optimal LP solution. We categorize the vertices into multiple groups based on the values assigned to them by the above LP solution. Next, we find a suitably large independent set within each group. The best among these independent sets is the output. We then argue that the output independent set is a $O(\sqrt{n})$ approximation to the optimal solution.

Our next set of results provide some evidence that it may be difficult to significantly improve the $O(\sqrt{n})$-approximation ratio. This includes proving integrality gaps and hardness results.

Let us first briefly discuss the integrality gap results. We show that the above LP has an integrality gap of $\sqrt{n}$, even for the case where both the input graphs are comparability graphs and all the weights are unit. We then consider a powerful strengthening of the LP, wherein a variable $x(u)$ is added for each vertex u, which indicates whether $x(u)$ is selected in the independent set or not. Then, for each clique C of $\widehat{G}$, we add the constraint $\sum_{u \in C} x(u) \leq 1$. Such an LP is also considered while designing the polynomial time algorithm for the MaxIS problem on perfect graphs. In the case of MaxCIS, this LP is of exponential size and it is not clear whether it can be solved in polynomial time; it seems difficult to construct separation oracles (even approximate ones). Nevertheless, we show that even this strong LP has an integrality gap of $n^{0.16}$ (even for the unweighted case over comparability graphs). This result is derived as a consequence of a combinatorial result regarding unions of comparability graphs, due to Dumitrescu and Tóth [5] (discussed earlier).

On the hardness front, it is known that the 2-MaxCIS problem on linear forests is APX-hard [1] (a linear forest is a graph in which each component is a path). Linear forests are bipartite graphs, which are in turn comparability graphs. It follows that 2-MaxCIS problem on comparability graphs cannot be approximated within a factor of $(1 + \epsilon)$, for some

$\epsilon > 0$. We show that the hardness gap produced by the above APX-hardness can be amplified (for the case of comparability graphs). We do this via the well-known approach of considering powers of graphs. The same graph powering technique is used to amplify hardness gap for the MaxIS (or the maximum clique problem). However, the standard graph product typically used for this purpose (see [10]) does not serve in our scenario. The reason is that, in our scenario, we need a graph product under which, if G is the union of two comparability graphs, then G^r is also the union of two comparability graphs. The standard graph product mentioned above may not satisfy this property. So, we consider a different graph product and obtain the amplification. (Dumitrescu and Tóth [5] also use a similar graph product for proving a combinatorial result on comparability graphs (see Theorem 5)). Using this approach, we prove that if NP $\neq$ P, then the 2-MaxCIS problem on comparability graphs cannot be approximated within any constant factor. Via a simple extension of this result, we also show that if NP $\not\subseteq$ DTIME$[n^{O(\log n)}]$, then the above problem cannot be approximated within a factor of $2^{\sqrt{\log n}}$. Here, a challenging open problem is to show that it is NP-hard to approximate within a factor of n^{ϵ}, for some $\epsilon > 0$.

2 Preliminaries

In this section, we discuss some concepts and notations used in the paper. We also briefly discuss some known algorithmic results about perfect graphs and their special cases.

Notation: For two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$, let $G_1 \oplus G_2$ denote their union graph. Let $G = (V, E)$ be a graph. For a subset of vertices $X \subseteq V$, let $G[X]$ denote the subgraph induced by X in G. Let $p : V \to \mathbf{R}$ be a weight or profit function that assigns a profit or weight $p(u)$ to each vertex $u \in V$. For a subset of vertices $X \subseteq V$, we use $p(X)$ to denote the sum of profits over X ($p(X) = \sum_{u \in X} p(u)$).

Basic Graph Parameters: Let $G = (V, E)$ be a graph. We shall use the following notations to refer to some basic graph properties: (i) Independence number $\alpha(G)$: cardinality of the maximum independent set in G; (ii) Clique number $\omega(G)$: cardinality of the maximum clique in G; (iii) Chromatic number $\chi(G)$: minimum number of colors needed to color the vertices of G so that no two adjacent vertices of G receive the same color.

Comparability Graphs: Comparability graphs capture the comparability relationships among elements of a partial ordered set. In this paper, we shall use directed acyclic graphs (DAG) to define them. Consider a DAG $D = (V, E')$. We say that a vertex u is an *ancestor* of a vertex v, if there is a path from u to v. A pair of vertices u and v are said to be *comparable*, if either u is an ancestor of v or v is an ancestor of u; otherwise, they are said to be *incomparable*. Construct a graph $G = (V, E)$; add an edge between every pair of comparable vertices. The graph G is said to be the comparability graph of D. Alternatively, G can be obtained by taking the transitive closure of D and ignoring the directionality of the edges. A graph is said to be a comparability graph, if it is the comparability graph of some DAG. McConnell and Spinrad [11] present a linear time algorithm for testing whether an input graph is a comparability graph; moreover, if the graph is a comparability graph, then their algorithm also outputs a DAG representation (or transitive orientation) in linear time.

Perfect Graphs: Notice that for any graph G, $\chi(G) \geq \omega(G)$. A graph $G = (V, E)$ is said to be *perfect*, if for every induced subgraph H, $\chi(H) = \omega(H)$. Chudnovsky et al. [4, 3] proved that whether an input graph is perfect can be tested in polynomial time.

3 Weighted 2-MaxCIS Problem on Perfect Graphs

In this section, we present a $O(\sqrt{n})$ approximation algorithm for the 2-MaxCIS problem on perfect graphs.

In our approximation algorithm, we will need a polynomial procedure for the tasks of finding maximum weight independent sets and cliques in perfect graphs. Grötschel et al. [8] present polynomial time algorithms for performing both the tasks.

▶ **Theorem 1** ([8]). *The weighted* MaxIS *and weighted* MaxClique *problems can be solved in polynomial time for perfect graphs.*

Let $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ be the two input perfect graphs over a vertex set V of size n and let $\widehat{G} = (V, \widehat{E}) = G_1 \oplus G_2$ be their union. Let the input profit function be $p : V \to \mathbf{R}$, where $p(u)$ is the profit of a vertex $u \in V$.

Our approximation algorithm is based on a crucial lemma discussed next. Consider a subset $X \subseteq V$. Let $\omega_1(X) = \omega(G_1[X])$ and $\omega_2(X) = \omega(G_2[X])$, be the size of the maximum cliques in the graph induced by X in G_1 and G_2, respectively. The lemma below presents a polynomial time algorithm for finding an independent set I contained in X and having reasonably large profit. The idea behind the lemma is as follows. Suppose $\widehat{H}$ is the union of two perfect graphs H_1 and H_2 over a vertex set of size m. Then, H_1 and H_2 can be colored in $\omega(H_1)$ colors and $\omega(H_2)$ colors, respectively. Therefore, their union $\widehat{H}$ can be colored using $\omega(H_1)\omega(H_2)$ colors. Hence, $\widehat{H}$ must have an independent set of size $m/(\omega(H_1)\omega(H_2))$. The lemma generalizes this idea to the weighted case and applies it to subgraphs. More importantly, it gives a polynomial time procedure for finding such a good independent set. Dumitrescu and Tóth [5] use a similar argument in the context of proving a certain combinatorial property about unions of perfect graphs.

▶ **Lemma 2.** *Let $X \subseteq V$. There exists a subset $I \subseteq X$ such that I is an independent set of $\widehat{G}$ and*

$$p(I) \geq \frac{p(X)}{\omega_1(X)\omega_2(X)}.$$

Moreover, such a set can be found in polynomial time.

Proof. Since G_1 is perfect, $G_1[X]$ is also perfect (by definition). Thus, $G_1[X]$ can be colored using $\omega_1(X)$ colors. This means that X can be partitioned into $\omega_1(X)$ many subsets (i.e., color classes), which are all independent in $G_1[X]$. One of these independent sets must have profit at least $p(X)/\omega_1(X)$. Apply the algorithm given in Theorem 1 and find the maximum profit independent set of $G_1[X]$; let this be I'. We have $p(I') \geq p(X)/\omega_1(X)$. We shall now focus on $G_2[I']$. The graph $G_2[X]$ can be colored with $\omega_2(X)$ colors and so, $G_2[I']$ can also be colored using $\omega_2(X)$ colors. It follows that $G_2[I']$ has an independent set having profit at least $p(I')/\omega_2(X)$. Apply the algorithm given in Theorem 1 and find the maximum profit independent set of $G_2[I']$; let this be I. We have $p(I) \geq p(I')/\omega_2(X)$. It follows that $p(I) \geq p(X)/(\omega_1(X)\omega_2(X))$. Notice that I is an independent set in both G_1 and G_2, and hence, it is an independent set in $\widehat{G}$. ◀

Our $O(\sqrt{n})$-approximation uses the LP rounding approach. We start the description of the algorithm by first discussing our LP formulation.

Linear Program: Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be the set of all cliques of G_1 and G_2, respectively. Notice that any independent set of $\widehat{G}$ can pick at most one vertex from any clique $C \in \mathcal{C}_1$ (or any $C \in \mathcal{C}_2$). We shall express these constraints in the linear program. For each vertex

$u \in V$, add a variable $x(u)$ that indicates whether u is included in the independent set or not. The LP is shown next:

$$\begin{aligned}
\max \sum_{u \in V} p(u)x(u) \qquad & \text{subject to} \\
\sum_{u \in C} x(u) \leq 1 \qquad & \text{for all } C \in \mathcal{C}_1 \\
\sum_{u \in C} x(u) \leq 1 \qquad & \text{for all } C \in \mathcal{C}_2 \\
0 \leq x(u) \leq 1 \qquad & \text{for all } u \in V
\end{aligned}$$

The above LP may have exponentially many constraints, since $|\mathcal{C}_1|$ and $|\mathcal{C}_2|$ could be exponential in n. Nevertheless, we can solve it using a separation oracle. Recall that a separation oracle is a procedure which takes as input a fractional assignment $\mathbf{x} : V \to [0, 1]$ and decides whether $\mathbf{x}$ is a feasible solution to the LP; moreover, if $\mathbf{x}$ is not a feasible solution, the procedure should output a constraint violated by $\mathbf{x}$. In our case, we can perform the above task in polynomial time using the algorithm given in Theorem 1. Taking $\mathbf{x}$ to be the weight or profit function, find the maximum profit clique in G_1 and G_2, using the above algorithm. If both these cliques have profit at most 1, then we know that $\mathbf{x}$ is a feasible solution. Otherwise, the constraint corresponding to the clique having profit greater than 1 is violated. Therefore, we can construct the required separation oracle. Given such an oracle, the ellipsoid method can solve the LP and find the optimal fraction solution (see [8]).

Let $\mathbf{x} : V \to [0, 1]$ be the optimal fractional solution to the LP. For a subset of vertices $X \subseteq V$, let $\mathrm{LP}^*(X) = \sum_{u \in X} p(u)\mathbf{x}(u)$. Let $\mathrm{LP}^* = \mathrm{LP}^*(V)$ denote the profit of the LP solution. Let $\mathrm{P}_{\max} = \max_{u \in V} p(u)$ be the maximum profit.

Rounding Algorithm: We now describe our rounding scheme. First, partition the vertex set V in to SML and LRG, where $\mathrm{SML} = \{u : 0 \leq \mathbf{x}(u) \leq 1/\sqrt{n}\}$ and $\mathrm{LRG} = \{u : 1/\sqrt{n} \leq \mathbf{x}(u) \leq 1\}$. Thus, $\mathrm{LP}^* = \mathrm{LP}^*(\mathrm{SML}) + \mathrm{LP}^*(\mathrm{LRG})$.

As we shall see, it is easy to handle the set SML. So, let us ignore SML for now, and focus on LRG. We geometrically divide the interval $[(1/\sqrt{n}), 1]$ into $\ell = (\log n)/2$ ranges and classify the vertices u in LRG based on the range into which $\mathbf{x}(u)$ falls. Namely, for $0 \leq i < \ell$, define

$$U_i = \{u : (1/\sqrt{n})2^i \leq \mathbf{x}(u) \leq (1/\sqrt{n})2^{i+1}\}.$$

Thus, $U_0, U_1, \ldots, U_{\ell-1}$ forms a partition of LRG.

The rounding algorithm is as follows. For $0 \leq j < \ell$, apply the algorithm given in Lemma 2 on the set U_j (taking $X = U_j$) and find an independent set I_j of $\widehat{G}$ such that

$$p(I_j) \geq \frac{p(U_j)}{\omega_1(U_j)\omega_2(U_j)}. \tag{1}$$

Then, among these ℓ independent sets, choose the one having the maximum profit. Let I^* be the chosen set. Finally, consider the following two options: (i) the singleton set containing the vertex having the maximum profit $\mathrm{P}_{\max}$; (ii) the set I^*. Between these options, output the one having the maximum profit. Let $\mathrm{I}_{\mathrm{alg}}$ be the set output by the above rounding scheme.

Analysis: We shall now analyze the rounding scheme. The goal is to show that our algorithm has an approximation ratio of $O(\sqrt{n})$. The following lemma presents the main claim of the analysis.

▶ **Lemma 3.** $\mathrm{LP}^*(\mathrm{LRG}) \leq (4\sqrt{n})p(I^*)$.

We shall prove the lemma shortly. Let us complete the analysis assuming the lemma. Recall that $\mathrm{LP}^* = \mathrm{LP}^*(\mathrm{SML}) + \mathrm{LP}^*(\mathrm{LRG})$. Observe that $\mathrm{LP}^*(\mathrm{SML}) \leq \sqrt{n}\mathrm{P}_{\max}$ and $p(\mathrm{I}_{\mathrm{alg}}) \geq \mathrm{P}_{\max}$. Hence, $\mathrm{LP}^*(\mathrm{SML}) \leq \sqrt{n} \cdot p(\mathrm{I}_{\mathrm{alg}})$. The lemma implies that $\mathrm{LP}^*(\mathrm{LRG}) \leq (4\sqrt{n})p(\mathrm{I}_{\mathrm{alg}})$. It follows that $\mathrm{LP}^* \leq (5\sqrt{n})p(\mathrm{I}_{\mathrm{alg}})$. Thus, our algorithm has $O(\sqrt{n})$ approximation ratio. We now proceed to prove the lemma. The claim given below is useful for this purpose.

▶ **Lemma 4.** *For any* $0 \leq j < \ell$, $\mathrm{LP}^*(U_j) \leq 2(\sqrt{n}/2^j)p(I_j)$.

Proof. Let $\beta = (1/\sqrt{n})2^j$. Write $U = U_j$ and $I = I_j$. Our goal is to show that $\mathrm{LP}^*(U) \leq 2(1/\beta)p(I)$. Let $\omega_{\min} = \min\{\omega_1(U), \omega_2(U)\}$ and let $\omega_{\max} = \max\{\omega_1(U), \omega_2(U)\}$.

Equation 1 shows that

$$p(I) \geq \frac{p(U)}{\omega_{\min}\omega_{\max}}. \tag{2}$$

Let us now derive a bound on $\mathrm{LP}^*(U)$. By the definition of U, we have that $\beta \leq \mathbf{x}(u) \leq 2\beta$. Therefore, $\mathrm{LP}^*(U) \leq (2\beta)p(U)$. There exists a subset $C \subseteq U$ which is a clique in $G_1[X]$ or $G_2[X]$ such that $|C| = \omega_{\max}$. The LP contains a constraint corresponding to this clique and hence, $\sum_{u \in C} \mathbf{x}(u) \leq 1$. Since every $u \in U$ satisfies $\mathbf{x}(u) \geq \beta$, we have that $\beta\omega_{\max} \leq 1$. Thus, $\beta \leq (1/\omega_{\max})$. Putting together, we get that

$$\mathrm{LP}^*(U) \leq \frac{2p(U)}{\omega_{\max}}. \tag{3}$$

Equations 2 and 3 imply that $\mathrm{LP}^*(U) \leq 2\omega_{\min}p(I)$. Recall that $\beta\omega_{\max} \leq 1$. Thus, $\omega_{\max} \leq (1/\beta)$ and hence, $\omega_{\min} \leq (1/\beta)$. We have proved the lemma. ◀

Proof of Lemma 3: We have $\mathrm{LP}^* = \sum_{j=0}^{\ell-1} \mathrm{LP}^*(U_j)$. Lemma 4 shows that for all U_j, $\mathrm{LP}^*(U_j) \leq 2(\sqrt{n}/2^j)p(I^*)$. Therefore,

$$\mathrm{LP}^* \leq (2\sqrt{n})p(I^*) \sum_{j=0}^{\ell-1} (1/2^j) \leq (4\sqrt{n})p(I^*).$$

◀

Our algorithm and analysis can easily be extended to the case of weighted k-MaxCIS problem on perfect graphs, for a constant k. For this problem, we get an algorithm with an approximation ratio of $O(n^{(k-1)/k})$.

4 Integrality Gaps

Here, we show that the LP considered in the previous section has an integrality gap of $\sqrt{n}$, even when the input graphs are unions of two comparability graphs and the input is unweighted. Then, we consider a strengthening of the LP and show that it has an integrality gap of $n^{0.16}$.

Fix any square number n. We shall construct a graph $G = (V, E)$ on n vertices such that G is the union of two comparability graphs and the LP has an integrality gap of $\sqrt{n}$ on G. Let $k = \sqrt{n}$. Let $A = \{1, 2, \ldots, k\}$ and $V = A \times A$. We will describe G by presenting two DAGs $D_1 = (V, E_1)$ and $D_2 = (V, E_2)$. The DAG D_1 is as follows. Consider a pair of vertices $u = \langle a_1, a_2 \rangle$ and $v = \langle b_1, b_2 \rangle$ belonging to V, where $a_1, a_2, b_1, b_2 \in A$. Add an edge from u to v, if $b_1 \geq a_1 + 1$. The DAG D_2 is constructed in a similar fashion. Add an edge from a vertex $u = \langle a_1, a_2 \rangle$ to a vertex $v = \langle b_1, b_2 \rangle$, if $b_2 \geq a_2 + 1$. The two DAGs can be visualized by considering a $k \times k$ grid of vertices. In D_1, an edge is drawn from a vertex u to all the vertices appearing in rows below the row of u; in D_2, an edge is drawn from a

vertex u to all the vertices appearing in columns to the right of column of u. It is easy to see that D_1 and D_2 are acyclic. Let G_1 and G_2 be the comparability graphs of D_1 and D_2, respectively. Take G to be the union of G_1 and G_2. Notice that G is the complete graph on n vertices and so, $\alpha(G) = 1$. On the other hand, $\omega(G_1) = k$ and $\omega(G_2) = k$. So, we get a feasible LP solution by setting $\mathbf{x}(u) = 1/k$, for all $u \in V$. This LP solution has profit $n/k = \sqrt{n}$. We conclude that the LP has an integrality gap of $\sqrt{n}$ on G.

We now describe a strengthening of the previous LP and exhibit integrality gap. The drawback with the previous LP is that it adds one constraint for each clique of G_1 and one constraint for each clique of G_2. A natural idea is to add a constraint for each clique of the union graph G. Namely, for each clique C of G, add a constraint $\sum_{u \in C} x(u) \leq 1$. Notice that this LP does not have any integrality gap on the graph instances of the previous construction. Nevertheless, we show that even this strengthened LP has an integrality gap of $n^{0.16}$. For this, we use a combinatorial result proved by Dumitrescu and Tóth [5].

▶ **Theorem 5** ([5]). *There exists an infinite family of graphs $\{G\}$ such that each graph G is the union of two comparability graphs and $\omega(G) \leq n^{0.42}$ and $\alpha(G) \leq n^{0.42}$, where n is the number of vertices in G.*

The integrality gap for the strengthened LP follows immediately from the above theorem. Consider any graph $G = (V, E)$ given by this theorem. For all $u \in V$, set $\mathbf{x}(u) = 1/n^{0.42}$. Notice that this is feasible solution to the strengthened LP. Its profit is $n/n^{0.42} = n^{0.58}$. On the other hand, the maximum independent set of G is of size at most $n^{0.42}$. It follows that the LP has an integrality gap of $n^{0.16}$.

5 2-MaxCIS : Hardness Results

Bar-Yehuda et al. [1] show that the 2-MaxCIS problem is APX-hard on linear forests (a linear forest is a graph in which each component is a path). They proved this result by showing that any 3-regular graph can be expressed as the union of two linear forests. Linear forests are bipartite graphs, which are in turn comparability graphs. It follows that the 2-MaxCIS problem on comparability graphs in APX-hard. This means that there exists an $\epsilon > 0$ such that it is NP-hard to approximate the problem within a factor of $(1 + \epsilon)$. The above APX-hardness proof can be transformed into a hardness gap, as stated in the theorem below. A similar claim for the MaxIS problem on 3-regular graphs is implicit in the work of Chlebík and Chlebíková [2]. We can derive the theorem below by combining their result with that that of Bar-Yehuda et al.

▶ **Theorem 6.** *There exists a polynomial time algorithm and a constant $\epsilon > 0$ with the following property. The algorithm takes as input a boolean formula φ and outputs two comparability graphs G_1 and G_2, and a number k such that*

$$\varphi \in \text{SAT} \implies \alpha(\widehat{G}) \geq (1 + \epsilon)k$$
$$\varphi \notin \text{SAT} \implies \alpha(\widehat{G}) \leq k,$$

where $\widehat{G} = G_1 \oplus G_2$.

We next amplify the gap produced by Theorem 6 and show that the MaxCIS problem on comparability graphs cannot be approximated within any constant factor. We shall perform the amplification by using the well-known approach of taking graph powers. Towards that goal, we define a graph power satisfying two important properties: (i) if G is the union of two comparability graphs, then G^r is also the union of two comparability graphs; (ii)

$\alpha(G^r) = [\alpha(G)]^r$. Dumitrescu and Tóth [5] also use a similar graph product for proving a combinatorial result (Theorem 5) on comparability graphs.

The graph power is described next. Let $G = (V, E)$ be any graph and let r be any integer. Construct a graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$ as follows. The vertex set $\widetilde{V}$ is given by $\widetilde{V} = V \times V \times \cdots V$, where the cartesian product is taken r times. The edges of $\widetilde{G}$ are described next. We add an edge between two vertices $\widetilde{u} = \langle u_1, u_2, \ldots, u_r \rangle$ and $\widetilde{v} = \langle v_1, v_2, \ldots, v_r \rangle$, if $(u_j, v_j) \in E$, where j is the smallest number such that $u_j \neq v_j$. We define the graph power G^r to be the graph $\widetilde{G}$ constructed above. The above graph powering has the following properties.

▶ **Lemma 7.** *Consider any integer r.*

- *For any graph G, $\alpha(G^r) = [\alpha(G)]^r$.*
- *If G is a comparability graph then G^r is also a comparability graph.*
- *Let $\widehat{G} = G_1 \oplus G_2$ be the union of two graphs G_1 and G_2. Then, $\widehat{G}^r = G_1^r \oplus G_2^r$.*

Proof. Consider the first claim. Let $\alpha(G) = k$ and let I be an independent set of G of size k. Let $I' = I \times I \times \cdots I$, where the cartesian product is taken r times. I' is an independent set in G^r and it is of size k^r. Thus, $\alpha(G^r) \geq k^r$. To see the reverse direction, consider an independent set I' of G^r. For $1 \leq j \leq r$, define

$$U_j = \{\langle u_1, u_2, \ldots, u_j \rangle : \text{there exists } u_{j+1}, u_{j+2} \ldots, u_r \text{ such that } \langle u_1, u_2, \ldots, u_r \rangle \in I'\}.$$

By induction on j, we shall show that for all $1 \leq j \leq r$, $|U_j| \leq k^j$. For the base case of $j = 0$, the set U_1 is an independent set in G and hence, $|U_1| \leq k$. To prove the induction step, consider the set U_{j+1}. Pick any $\langle u_1, u_2, \ldots, u_j \rangle \in U_j$. The set $\{u : \langle u_1, u_2, \ldots, u_j, u \rangle \in U_{j+1}\}$ is an independent set in G and so, its cardinality is at most k. By induction $|U_j| \leq k^j$ and hence, $|U_{j+1}| \leq k^{j+1}$. It follows that $I' = U_r$ has cardinality at most k^r.

Consider the second claim. Let $G = (V, E)$ be the comparability graph of some DAG $D = (V, E')$. Without loss of generality assume that D is transitively closed (i.e., if $(a, b) \in E'$ and $(b, c) \in E'$ then $(a, c) \in E'$). We shall define a directed graph version of our graph power and show that G is the comparability graph of D^r. Construct a directed graph $\widetilde{D} = (\widetilde{V}, \widetilde{E})$ as follows. Define $\widetilde{V} = V \times V \times \cdots V$, where the cartesian product is taken r times. Add an edge from a vertex $\widetilde{u} = \langle u_1, u_2, \ldots, u_r \rangle$ to the vertex $\widetilde{v} = \langle v_1, v_2, \ldots, v_r \rangle$, if $(u_j, v_j) \in E'$, where j is the smallest number such that $u_j \neq v_j$. Let $D^r = \widetilde{D}$. We first claim that the graph $\widetilde{D}$ constructed above is transitively closed. To see this, consider three vertices $\widetilde{x} = \langle x_1, x_2, \ldots, x_r \rangle$, $\widetilde{y} = \langle y_1, y_2, \ldots, y_r \rangle$ and $\widetilde{z} = \langle z_1, z_2, \ldots, z_r \rangle$ such that $(\widetilde{x}, \widetilde{y}) \in \widetilde{E}$ and $(\widetilde{y}, \widetilde{z}) \in \widetilde{E}$. Let j_1 and j_2 be the smallest integers such that $x_{j_1} \neq y_{j_1}$ and $y_{j_2} \neq z_{j_2}$, respectively. Consider the three cases of $j_1 < j_2$, $j_2 < j_1$ and $j_1 = j_2$. In the first case, j_1 is the smallest number such that $x_{j_1} \neq z_{j_1}$. For this index j_1, we have that $(x_{j_1}, y_{j_1}) \in E'$ and $y_{j_1} = z_{j_1}$, and hence, $(x_{j_1}, z_{j_1}) \in E'$. It follows that $(\widetilde{x}, \widetilde{z}) \in \widetilde{E}$. The second case is handled in a similar fashion. For the third case, let $j = j_1 = j_2$. Notice that $(x_j, y_j) \in E'$ and $(y_j, z_j) \in E'$ and hence, $(x_j, z_j) \in E'$ (since D' is transitively closed). It follows that $(\widetilde{x}, \widetilde{z}) \in \widetilde{E}$. We have shown that $\widetilde{D}$ is transitively closed. This also shows that $\widetilde{D}$ is acyclic. We can now easily argue that G is the comparability graph of $\widetilde{D}$.

The third claim is easy to see. ◀

Using the graph power and Lemma 7, we next argue that it is NP-hard to approximate the 2-MaxCIS problem on comparability graphs within any constant factor. Theorem 6 provides an algorithm that produces a gap of $(1 + \epsilon)$. We can amplify the gap by graph powering, as described next. Fix any constant r. Let the algorithm given by Theorem 6 be denoted as $\mathcal{A}$. Consider an algorithm $\mathcal{B}$ that takes as input a formula φ and outputs graphs G_1^r and G_2^r, where G_1 and G_2 are the graphs output by $\mathcal{A}$ on input φ. Lemma 7 shows that

G_1^r and G_2^r are comparability graphs. The same lemma shows that $\alpha(G') = [\alpha(\widehat{G})]^r$, where $G' = G_1^r \oplus G_2^r$ and $\widehat{G} = G_1 \oplus G_2$. It follows that

$$\varphi \in \text{SAT} \implies \alpha(G') \geq (1+\epsilon)^r k^r$$
$$\varphi \notin \text{SAT} \implies \alpha(G') \leq k^r.$$

The algorithm $\mathcal{B}$ runs in polynomial time and produces a gap of $(1+\epsilon)^r$. Consider any constant c and set $r = (\log c)/\log(1+\epsilon)$. Then, the above algorithm with parameter r produces a gap of c. The above argument leads to the following theorem.

▶ **Theorem 8.** *If* NP $\neq$ P *then the* MaxCIS *problem on comparability graphs cannot be approximated within any constant factor.*

We can amplify the gap further. Instead of fixing r to be a constant, let us set $r = d \log n$, where n is the number of vertices in $\widehat{G}$ and d is a suitably defined constant. The output graph $\widehat{G}^r$ will have $N = n^r$ vertices and the gap would become $(1+\epsilon)^{d \log n}$. Setting $d = (1/\log(1+\epsilon))^2$, we get a gap of $2^{\sqrt{N}}$. However, taking the rth power would take $n^{O(\log n)}$ time. The above construction leads to the following result.

▶ **Theorem 9.** *If* NP $\not\subseteq$ DTIME$[n^{O(\log n)}]$*, then* MaxCIS *problem on comparability graphs cannot be approximated within a factor of* $2^{\sqrt{\log n}}$.

References

1. R. Bar-Yehuda, M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. *SIAM Journal of Computing*, 36(1):1–15, 2006.
2. M. Chlebík and J. Chlebíková. Complexity of approximating bounded variants of optimization problems. *Theoretical Computer Science*, 354(3):320–338, 2006.
3. M. Chudnovsky, G. Cornuéjols, X. Liu, P. Seymour, and K. Vuskovic. Recognizing Berge graphs. *Combinatorica*, 25(2):143–186, 2005.
4. M. Chudnovsky, N. Robertson, P. Seymour, and R.Thomas. The strong perfect graph theorem. *Annals of Mathematics*, 164(1):51–229, 2006.
5. A. Dumitrescu and G. Tóth. Ramsey-type results for unions of comparability graphs. *Graphs and Combinatorics*, 18(2):245–251, 2002.
6. A. Frank. Some polynomial algorithms for certain graphs and hypergraphs. In *5th British Combinatorial Conference, University of Aberdeen, Aberdeen*, page 211–226, 1975.
7. F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM Journal of Compututing*, 1(2):180–187, 1972.
8. M. Grötschel, L. Lovász, and A. Schrijver. The ellipsoid method and its consequences in combinatorial optimization. *Combinatorica*, 1(2):169–197, 1981.
9. D. Hermelin and D. Rawitz. Optimization problems in multiple subtree graphs. In *7th WAOA, LNCS 5893*, pages 194–204, 2009.
10. D. Hochbaum, editor. *Approximation algorithms for NP-hard problems.* PWS Publishing Co., Boston, 1997.
11. R. McConnell and J. Spinrad. Linear-time transitive orientation. In *Proceedings of the Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 19–25, New Orleans, Louisiana, 1997.
12. R. Möhring. Algorithmic aspects of comparability graphs and interval graphs. In I. Rival, editor, *Graphs and orders*, pages 41–101. D. Reidel, Boston, 1985.
13. D. Zuckerman. Linear degree extractors and the inapproximability of max-clique and chromatic number. *Theory of Computing*, 3(1):103–128, 2007.

Fast equivalence-checking for normed context-free processes *

Wojciech Czerwiński and Sławomir Lasota

Institute of Informatics, University of Warsaw
Banacha 2, Warsaw, Poland
wczerwin,sl@mimuw.edu.pl

Abstract

Bisimulation equivalence is decidable in polynomial time over normed graphs generated by a context-free grammar. We present a new algorithm, working in time $\mathcal{O}(n^5)$, thus improving the previously known complexity $\mathcal{O}(n^8\text{polylog}(n))$. It also improves the previously known complexity $\mathcal{O}(n^6\text{polylog}(n))$ of the equality problem for simple grammars.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.260

1 Introduction

Equivalence checking, that is determining whether two systems are equal under a given notion of equivalence, is an important verification problem with a long history. In this paper we consider systems described by context-free grammars. It is well known that language equivalence is undecidable in this class [1]. A decidability result was obtained by Korenjak and Hopcroft [12] for a restricted class of deterministic context-free grammars (*simple grammars*). Remarkably, the language containment is undecidable even for simple grammars [7].

In the context of process algebras, a grammar may be considered as a description of a transition graph rather than a language. The adequate concept of equivalence is then *bisimilarity* (bisimulation equivalence), a notion strictly finer than language equivalence. For graphs generated by context-free grammars, called *context-free processes*, bisimilarity is known to be decidable due to the result of [5]. It has also been demonstrated that bisimilarity is the only equivalence in van Glabbeek's spectrum [8] which is decidable for context-free processes. This places bisimilarity in a very favourable position.

Historically the first decision procedure for bisimilarity on infinite-state systems was given by [3] for a class of *normed* context-free processes, those defined by grammars in which, roughly, each nonterminal generates at least one word. Clearly, language equivalence is still undecidable in this class, as normedness assumption does not facilitate testing language equality. As language equivalence and bisimilarity coincide on deterministic graphs the result of [3] was a strict extension of [12]. Later, decidability was extended to all context-free processes [5].

In the normed case, a series of consecutive papers [4, 10, 11] lead finally to a remarkable polynomial-time algorithm of [9] for bisimilarity. The working time $\mathcal{O}(n^{13})$ was however not satisfactory and hence further results followed: an $\mathcal{O}(n^7 \mathit{polylog}\ n)$-time algorithm was proposed for equivalence of simple grammars [2] and an $\mathcal{O}(n^8 \mathit{polylog}\ n)$-time algorithm was given for bisimilarity [13]. The latter cut down to simple grammars works in time $\mathcal{O}(n^6 \mathit{polylog}\ n)$. We report a further progress: we give an $\mathcal{O}(n^5)$ time algorithm for bisimi-

* This paper is partially supported by Polish government grant no. N N206 356036 and by EU-FET-IP *Software Engineering for Service-Oriented Overlay Computers* SENSORIA.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 260–271

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

larity on normed context-free processes, thus improving the previously known bound by a factor of $\mathcal{O}(n^3)$. We improve the case of simple grammars just as well.

Our approach is substantially different from all recent algorithms which, roughly, compute the base of the bisimulation equivalence by eliminating the incorrect decompositions from the initial 'overapproximating' base. Instead, we apply the refinement scheme used previously for the *commutative* context-free processes. Our starting point is the algorithm, given in [6], that works for both normed commutative and non-commutative processes. Intuitively, the algorithm combines the algorithmic theory of compressed strings, useful for non-commutative case, with the iterative approximation refinement scheme used in commutative case. In this paper we demonstrate, roughly speaking, that the latter scheme may be implemented efficiently for non-commutative processes in time $\mathcal{O}(n^5)$.

We start by defining the problem in Section 1. The following section introduces a necessary background material, namely pattern-matching in compressed strings and unique decomposition in a finitely-generated monoid, and explains the refinement scheme exploited in our algorithm. Then, in Sections 3 and 4 we present the algorithm itself. Section 3 provides a general outline and intentionally omits a number of details; in Section 4 we provide all the implementation details missing in Section 3.

Context-Free Processes and Bisimilarity.

Ingredients of a *process definition* Δ are a finite alphabet Σ, a finite set $\mathtt{V}$ of variables, and a finite set of rules

$$X \stackrel{a}{\longrightarrow} \alpha, \tag{1}$$

with $a \in \Sigma$ and $\alpha \in \mathtt{V}^*$. Such process definitions are usually called in the literature *Basic Process Algebra*, or *Context-Free Processes*. The explanation of the latter is that each rule can be seen as a production $X \longrightarrow a\alpha$ of a context-free grammar in Greibach normal form. Elements of $\mathtt{V}^*$ are called here *processes*; a variable X can be seen as an elementary process.

Δ defines a transition system: its states are processes $\alpha \in \mathtt{V}^*$; and for each $a \in \Sigma$, there is a transition relation containing triples (α, a, β), where $a \in \Sigma$ and $\alpha, \beta \in \mathtt{V}^*$, written $\alpha \stackrel{a}{\longrightarrow} \beta$. The transition relations are defined by a prefix rewriting: $X\beta \stackrel{a}{\longrightarrow} \alpha\beta$ whenever Δ contains a rule $X \stackrel{a}{\longrightarrow} \alpha$, and $\beta \in \mathtt{V}^*$.

▶ **Definition 1.** Given a binary relation R over $\mathtt{V}^*$, we say that a pair (α, β) of processes satisfies expansion in R, written $(\alpha, \beta) \in \exp(R)$, if

- whenever $\alpha \stackrel{a}{\longrightarrow} \alpha'$, there exists some β' with $\beta \stackrel{a}{\longrightarrow} \beta'$ and $(\alpha', \beta') \in R$; and
- whenever $\beta \stackrel{a}{\longrightarrow} \beta'$, there exists some α' with $\alpha \stackrel{a}{\longrightarrow} \alpha'$ and $(\alpha', \beta') \in R$.

A binary relation S satisfies expansion in R if each pair $(\alpha, \beta) \in S$ does, i.e., $S \subseteq \exp(R)$. A relation R is a *bisimulation* if it satisfies expansion in itself. We say that α and β are *bisimilar*, denoted by $\alpha \sim \beta$, if (α, β) belongs to some bisimulation.

From now on we assume that Δ is *normed*, i.e., for each variable $X \in \mathtt{V}^*$ there is a finite sequence $X \stackrel{a_1}{\longrightarrow} \alpha_1 \ldots \stackrel{a_k}{\longrightarrow} \alpha_k = \varepsilon$ leading from X to the empty process ε. By $|X|$ denote the smallest length of such sequence and call it the *norm* of X (intuitively, $|X|$ is the length of the shortest word generated from X).

We consider the following Normed-BPA-Bisim Problem:

Instance: A normed process definition Δ and two variables $X, Y \in \mathtt{V}$.
Question: Is $X \sim Y$?

A more general problem of checking whether $\alpha \sim \beta$, for given $\alpha, \beta \in \mathtt{V}^*$, can be easily reduced to the above one. The size of Δ, denoted by N, is the sum of lengths of all the rules in Δ. Clearly $n \leq N$. Our main result is the following:

▶ **Theorem 2.** Normed-BPA-Bisim Problem *can be solved in time* $\mathcal{O}(N^5)$.

Thinking of Δ as of a grammar, call Δ a *simple grammar* if for each X and a, there is at most one rule (1) in Δ. As a direct corollary of Theorem 2 we obtain:

▶ **Corollary 3.** *Equivalence of simple grammars can be solved in time* $\mathcal{O}(N^5)$.

Our algorithm, similarly to previous ones [9, 13], builds a finite *base* of $\sim$ that, once constructed, allows to answer the Normed-BPA-Bisim Problem in constant time.

2 Preliminaries

Acyclic morphisms.

Let $\mathtt{A}$ be a finite set of terminal symbols, ranged over by $a, b, \ldots$, and let $\mathtt{S}$ be a finite set of non-terminal symbols, ranged over by $x, y, z, \ldots$. Assume a total ordering $<$ of non-terminal symbols. An *acyclic morphism* is a mapping $h : \mathtt{S} \to (\mathtt{S} \cup \mathtt{A})^*$ such that all symbols appearing in $h(x)$ are strictly smaller than x wrt. $<$. We implicitely extend the domain of h to $\mathtt{S} \cup \mathtt{A}$, assuming h to be identity on $\mathtt{A}$. Due to the acyclicity requirement, h induces a monoid morphism $h^* : (\mathtt{S} \cup \mathtt{A})^* \to \mathtt{A}^*$, as the limit of compositions $h, h^2 = h \circ h, \ldots$. Formally, $h^*(z) = h^k(z)$, for the smallest k with $h^k(z) \in \mathtt{A}^*$. Then the extension of h^* to all strings in $(\mathtt{S} \cup \mathtt{A})^*$ is as usual. Therefore each symbol $z \in \mathtt{S}$ *represents* a nonempty string $h^*(z)$ over A. Its length $\|h^*(z)\|$ may be exponentially larger than the size of h, written $\mathtt{size}(h)$, defined as the sum of lengths of all strings $h(z)$, $z \in \mathtt{S}$.

A relevant parameter of a symbol z, wrt. an acyclic morphism h, is its *depth*, written $\mathtt{depth}_h(z)$, and defined as the longest path in the derivation tree of z. A depth of h, written $\mathtt{depth}(h)$, is the greatest depth of a symbol.

An acyclic morphism h is *binary* if $\|h(z_i)\| = 2$, for all $z_i \in \mathtt{S}$. Any acyclic morphism h may be transformed to the equivalent binary one: replace each $h(z_i)$ of length greater than 2 with a balanced binary tree, using $\|h(z_i)\| - 2$ auxiliary symbols. Note that in consequence the depth of h may increase by a logarithmic factor, but the size of h may only increase by a constant factor. In the sequel we only consider binary morphisms.

In a word of length n we distinguish $n + 1$ *positions* $0 \ldots n$. If $h(z) = xy$, i.e., $h^*(z) = h^*(x)\, h^*(y)$, then by *the cutting position* in z we mean the position in $h^*(z)$ equal to the length of $h^*(x)$. We say that a substring *touches* a given position if this position is either inside this substring or on the border. The *occurrence table of* h stores, for each two symbols $x, y \in \mathtt{S}$, the set of starting positions of occurrences of $h^*(x)$ in $h^*(y)$ that touch the cutting position in y. The whole table may be stored compactly in $\mathcal{O}(|\mathtt{S}|^2)$ memory due to the following:

▶ **Lemma 4.** *(Basic Lemma [15]) The set of starting positions of occurences of* $h^*(x)$ *in* $h^*(y)$ *that touch the cutting position in* y*, if non-empty, is an arithmetic progression.*

▶ **Theorem 5.** *([14]) Given a binary acyclic morphism* h*, one may compute in time* $\mathcal{O}(\mathtt{size}(h)^2 \cdot \mathtt{depth}(h))$ *the occurence table of* h.

Generalized acyclic morphisms.

From now on assume that each terminal symbol $a \in \mathtt{A}$ has assigned its norm $|a|$, a positive integer. Norm extends additively to all strings from $\mathtt{A}^*$. For non-terminal symbols, we put $|z| := |h^*(z)|$.

▶ **Lemma 6.** *Given a binary acyclic morphism h, a symbol $z \in \mathtt{S}$, and $k < \|h^*(z)\|$, one may compute in time $\mathcal{O}(\mathtt{depth}_h(z))$ an acyclic morphism h' extending h, such that one of new symbols of h' represents the suffix of $h^*(z)$ of norm k (assumed that such exists), and $\mathtt{size}(h') \leq \mathtt{size}(h) + \mathcal{O}(\mathtt{depth}_h(z))$ and $\mathtt{depth}(h') = \mathtt{depth}(h)$.*

For efficiency reasons it is favourable *not* to compute explicitely the representation of the suffix of $h^*(z)$ of norm k. Instead, we will represent this suffix symbolically, as follows. By now, an acyclic morphism was defined by equations of the form $h(z) = x\,y$, meaning $h^*(z) = h^*(x)\,h^*(y)$. Now we will allow a more general form:

$$h(z) = x\,\mathrm{suffix}_k(y), \tag{2}$$

where $0 < k \leq |y|$, to mean that $h^*(z)$ is concatenation of $h^*(x)$ and the suffix of $h^*(y)$ of norm k. Note that (2) is well defined only when the required suffix exists. As a particular case, for $k = |y|$, one gets the standard definition $h(z) = x\,y$. As before we asume acyclicity in (2), i.e., $x, y < z$.

Theorem 5 may be adapted to the generalized acyclic morphisms, with a slightly worst time. Naively, one could get rid of all 'truncated' variables y in (2) using Lemma 6, ending with a quadratic blow-up of the size of the morphism, and then apply Theorem 5. We claim that one can do better:

▶ **Theorem 7.** *Given a generalized binary acyclic morphism h, one may compute in time $\mathcal{O}(\mathtt{size}(h)^3 \cdot \mathtt{depth}(h))$ the occurence table of h.*

From now on generalized acyclic morphisms are briefly called acyclic morphisms.

Unique decomposition.

Assume from now on a fixed normed process definition Δ, i.e., a finite alphabet Σ, a finite set $\mathtt{V} = \{X_1, \ldots, X_n\}$ of variables, and a finite set of rules of the form $X_i \xrightarrow{a} \alpha$, $a \in \Sigma, \alpha \in \mathtt{V}^*$. The complexity considerations in this section and later on are wrt. the size N of Δ.

Assume also that variables $\mathtt{V} = \{X_1, \ldots, X_n\}$ of a process definition are ordered according to non-decreasing norm: $|X_i| \leq |X_j|$ whenever $i < j$. We write $X_i < X_j$ if $i < j$. Note that $|X_1|$ is necessarily 1, and that norm of a variable is at most exponential wrt. the size of Δ, understood as the sum of lengths of all rules.

A congruence is *norm-preserving* if whenever α and β are related then $|\alpha| = |\beta|$. Let $\equiv$ be an arbitrary norm-preserving congruence in $\mathtt{V}^*$. Intuitively, an elementary process X_i is *decomposable* if $X_i \equiv \alpha\beta$ for some $\alpha, \beta \neq \epsilon$. Note that $|\alpha|, |\beta| < |X_i|$ then. For technical convenience we prefer to apply a slightly different definition. We say that X_i is *decomposable* wrt. $\equiv$, if $X_i \equiv \alpha$ for some process $\alpha \in \{X_1, \ldots, X_{i-1}\}^*$; otherwise, X_i is called *indecomposable*, or *prime* wrt. $\equiv$. In particular, X_1 is always prime.

Denote by $\mathtt{P}$ the set of primes wrt. $\equiv$. It is easy to show by induction on norm that for each process α there is some $\gamma \in \mathtt{P}^*$ with $\alpha \equiv \gamma$; in such case γ is called a *prime decomposition* of α. Note that a prime decomposition of X_i is either X_i itself, or it belongs to $\{X_1, \ldots, X_{i-1}\}^*$. We say that $\equiv$ has the *unique decomposition property* if each process has precisely one prime

decomposition. While the set P of primes depends on the chosen ordering of variables (in case $X_i \equiv X_j$, $i \neq j$), the unique decomposition property does not.

The following lemma is shown by considering the unique prime decompositions:

▶ **Lemma 8** (Left cancellation). *If $\equiv$ has the unique decomposition property and $\gamma\alpha \equiv \gamma\beta$ then $\alpha \equiv \beta$.*

The refinement step.

A transition $\alpha \xrightarrow{a} \beta$ is called *norm-reducing* if $|\beta| < |\alpha|$; we write $\alpha \xrightarrow{a}_{\text{n-r}} \beta$ in such case. We will need a concept of norm-reducing bisimulation (n-r-bisimulation, in short), i.e., a bisimulation over the transition system restricted to only norm-reducing transitions. The appropriate norm-reducing expansion wrt. R (cf. Def. 1) will be written as n-r-exp(R). Every bisimulation is a n-r-bisimulation (as a norm-reducing transition must be matched in a bisimulation by a norm-reducing one) but the converse does not hold in general.

▶ **Proposition 1.** *Each n-r-bisimulation, and hence each bisimulation, is norm-preserving.*

For a norm-preserving equivalence $\equiv$ over processes, let $\sim^{\equiv}_{\text{n-r}}$ denote the union of all n-r-bisimulations contained in $\equiv$. It witnesses most of typical properties of bisimulation equivalence. Being the union of n-r-bisimulations, $\sim^{\equiv}_{\text{n-r}}$ is a n-r-bisimulation itself, in fact the greatest n-r-bisimulation that is contained in $\equiv$. It admits the following fix-point characterization:

▶ **Proposition 2.** $(\alpha, \beta) \in \sim^{\equiv}_{n\text{-}r}$ *iff* $\alpha \equiv \beta$ *and* $(\alpha, \beta) \in$ *n-r-exp*$(\sim^{\equiv}_{n\text{-}r})$.

Moreover $\sim^{\equiv}_{\text{n-r}}$ is clearly an equivalence as $\equiv$ is assumed to be so. The relation $\sim^{\equiv}_{\text{n-r}}$ may be thus seen as the bisimulation equivalence relativized to pairs of processes related by $\equiv$ and to norm-reducing moves only.

The relativized bisimulation equivalence will play a crucial role in the algorithm, that will work by consecutive refinements of a current congruence until it finally stabilizes. Instead of the classical refinement step $\equiv \mapsto \equiv \cap \exp(\equiv)$, we prefer to use the following one:

$$\equiv \;\mapsto\; \sim^{\equiv \cap \exp(\equiv)}_{\text{n-r}} .$$

This transformation will be refered to as *the refinement step*, and $\sim^{\equiv \cap \exp(\equiv)}_{\text{n-r}}$ will be called *the refinement of* $\equiv$.

By the results of [6] specialized to normed BPA, it follows:

▶ **Lemma 9.** *([6]) If a norm-preserving congruence $\equiv$ has the unique decomposition property then the refinement of $\equiv$ is a congruence with the unique decomposition property.*

3 Outline of the algorithm

Overall idea.

We describe the algorithm in a top-down manner, introducing the implementation details incrementally. The overall idea is as follows: we start with the initial congruence $\equiv$, given simply by the norm equality (cf. Prop. 1), and then perform the fixpoint computation by refining $\equiv$ until it finally stabilizes:

Initialize $\equiv$ as the norm equality.
REPEAT
replace $\equiv$ by its refinement
UNTIL $\equiv$ coincides with its refinement.

Note that the approximating congruence $\equiv$ always subsumes bisimulation equivalence $\sim$ in the course of the algorithm, $\sim \subseteq \equiv$. Moreover, if $\equiv$ and its refinement coincide, then $\equiv \subseteq \exp(\equiv)$ and thus the opposite implication $\equiv \subseteq \sim$ follows. Thus the approximation scheme is correct wrt. the bisimulation equivalence. At the end of this section we will argue that the algorithm always terminates after at most n iterations.

Now we will outline a way of implementing the scheme above.

Representation by an acyclic morphism.

Clearly, instead of the whole (infinite!) congruence $\equiv$, the algorithm should maintain a finite representation of $\equiv$. As a succint representation we choose an acyclic morphism $h : \mathtt{S} \to (\mathtt{S} \cup \mathtt{A})^*$. The set $\mathtt{A}$ of terminal symbols will consist of all variables X_i that are currently prime wrt. $\equiv$, and the set of non-terminal symbols $\mathtt{S}$ will contain the variables currently decomposable wrt. $\equiv$, together with some other auxiliary symbols, to be defined later on. We assume that the ordering $<$ on $\mathtt{S}$ is consistent with the ordering $X_1 < \ldots < X_n$. For any variable X_i, $h^*(X_i) \in \mathtt{A}^* \subseteq \mathtt{V}^*$ will describe the prime decomposition of X_i. The morphism h will represent $\equiv$ in the following sense: $\alpha \equiv \beta \iff h^*(\alpha) = h^*(\beta)$. Below we prefer to write $=_h$ instead of $\equiv$ to emphesize the role of h.

Recall that due to Lemma 9 the congruence $\equiv$ invariantly has the unique decomposition property, hence always some h exists that represents $\equiv$ (e.g., take as $h(X_i)$ the prime decomposition of X_i). Note however that the same congruence may be represented by many different acyclic morphisms. A key ingredient of the algorithm will be an efficient construction of a sufficiently succint one.

As the congruence $\equiv$ is norm-preserving, norm of X_i is always equal to norm of $h(X_i)$, and hence to norm of $h^*(X_i)$ as well.

Leftmost prime factors.

If $X_i \equiv X_j\gamma$, $j < i$ and X_j is prime wrt. $\equiv$ we say that X_j is the *leftmost prime factor* of X_i wrt. $\equiv$. Note that due to the unique decomposition property of $\equiv$, X_i may have at most one leftmost prime factor wrt. $\equiv$. Moreover, Lemma 8 guarantees that if $X_i \equiv X_j\alpha$ and $X_i \equiv X_j\beta$ are two decompositions of X_i, starting with the same variable X_j, then $\alpha \equiv \beta$ and thus $\alpha \equiv \beta$ is determined uniquely up to $\equiv$. In consequence, it is crucial just to know, for each variable X_i, which variable X_j, $j < i$, if any, is the leftmost prime factor of X_i wrt. the current congruence $\equiv$. In the algorithm, this information will be maintained using the indices $\mathbf{lpf}(i) \in \{1 \ldots n-1\}$, for $i \in \{2 \ldots n\}$, with the following meaning: if the variable X_i is currently decomposable wrt. $\equiv$, then $X_{\mathbf{lpf}(i)}$ is the leftmost prime factor of X_i. Clearly

$$\mathbf{lpf}(i) < i. \tag{3}$$

As $\equiv$ is represented by an acyclic morphism h, $X_{\mathbf{lpf}(i)}$ always belongs to $\mathtt{A}$ and is the first letter in $h^*(X_i)$ (and the first letter in $h(X_i)$ as well, which will become apparent shortly).

Outline of the algorithm.

```
FOR i ∈ {2 ... n} DO let lpf(i) := 1;
A := {X_1};  A' := A;
initialize h;

REPEAT
    FOR X_i ∈ {X_2 ... X_n} \ A DO                                        (*)
        IF ¬ lpfactor(X_i, X_lpf(i)) THEN                                 (**)
            FOR X_j ∈ {X_lpf(i)+1 ... X_i-1} ∩ (A' \ A) DO                (***)
                IF lpfactor(X_i, X_j) THEN
                    let lpf(i) := j;
                    ( h'(X_i) is defined as a side-effect of lpfactor(X_i, X_j) )
                    break the inner for loop;
                FI
            IF the inner for loop not broken THEN add X_i to A' FI
        FI
    A := A'; h := h';
UNTIL A does not change
```

In the FOR loops $(*)$ and $(***)$, the variables are assumed to be inspected in the increasing order $X_1 \leq \ldots \leq X_n$.

Each iteration of the REPEAT loop, as outlined above, corresponds to a single refinement step of $\equiv$: given the current acyclic morphism h it computes a new acyclic morphism, say h', representing the refinement of $=_h$ (the latter has unique decomposition property by Lemma 9). The subroutine $\mathbf{lpfactor}(X_i, X_j)$, invoked several times in the algorithm, is to check whether X_j is the leftmost prime factor of X_i wrt. the refinement of $=_h$. The acyclic morphism h' is computed as a side-effect of the invocations of **lpfactor**; thus $\mathbf{lpfactor}(X_i, X_j)$ is invoked under assumption that the value of h' is already known for the variables from $\{X_1 \ldots X_{i-1}\}$. Description of this subroutine and other implementation details related to the computation of h' are deferred to the next section. Here, we merely focus on the scheme of updating the indices $\mathbf{lpf}(i)$.

In the inner FOR loop $(***)$, the variable X_j ranges over $\{X_{\mathbf{lpf}(i)+1} \ldots X_{i-1}\} \cap (\mathtt{A}' \setminus \mathtt{A})$. The rationale behind restricting this range so is the following.

As the refinement of $=_h$ is clearly finer than $=_h$ (we may write $=_{h'} \subseteq =_h$), a decomposition wrt. the refinement of $=_h$ is automatically a decomposition wrt. $=_h$. Thus, if some X_i is decomposable wrt. $=_h$, then there are just two possibilities for a new value of $\mathbf{lpf}(i)$ (denote it by $\mathbf{lpf}(i)'$): either it remains unchanged, $\mathbf{lpf}(i)' = \mathbf{lpf}(i)$ (if the **lpfactor** $(**)$ succeeds), or it changes. In the latter case, its new value $\mathbf{lpf}(i)'$ may be only chosen from $\mathtt{A}' \setminus \mathtt{A}$, as otherwise X_i would have two different leftmost prime factors wrt. (the coarser) $=_h$. Moreover, the new value of $\mathbf{lpf}(i)$ must be necessarily bigger than the old one. This follows from the observation that the 'fresh' prime variable $X_{\mathbf{lpf}(i)'} \in \mathtt{A}' \setminus \mathtt{A}$ was not so in the previous iteration, and again by the unique decomposition property its leftmost prime factor was necessarily the same as the previous leftmost prime factor of X_i:

$$\mathbf{lpf}(\mathbf{lpf}(i)') = \mathbf{lpf}(i). \tag{4}$$

Thus $\mathbf{lpf}(i) < \mathbf{lpf}(i)'$ by (3).

The size of the set $\mathtt{A}$ of terminal symbols (containing exactly the prime variables wrt. $=_h$) increases in each iteration of the REPEAT loop (except for the very last iteration). Thus the total number of iterations is at most n.

Concerning the correctness: if $\mathtt{A}$ does not change in one iteration of the REPEAT loop, it clearly follows that $=_h$ coincides with its refinement, which guarantees that $=_h$ reaches the bisimulation equivalence, as discussed in the beginning of this section.

4 Implementation of the algorithm

Now we explain how the acyclic morphism h is initialized and refined during one iteration of the REPEAT loop; and how the subroutine **lpfactor** is implemented. Total running time is discussed at the end of this section.

Initialization of h.

For any right-hand side β of a production in Δ we introduce a non-terminal symbol Y_β, and define $h(Y_\beta) = \beta$. Then we transform h into binary form, if necesssary. This will possibly introduce further auxiliary symbols; in the sequel these further symbols will not be mentioned explicitly. The symbols Y_β (together with the other auxiliary symbols) will be continuously contained in $\mathtt{S}$ during the algorithm and their definition will never change. The number of the symbols is $\mathcal{O}(N)$.

Distinguish one fixed *norm-reducing* transition rule

$$X_i \xrightarrow{a_i}_{\text{n-r}} \alpha_i \tag{5}$$

for every variable X_i, $2 \leq i \leq n$; clearly $|X_i| = |\alpha_i| + 1$. These distinguished rules will be fixed in the algorithm. For $i \geq 2$, we initilize h by

$$h(X_i) = X_1\, Y_{\alpha_i}, \tag{6}$$

which gives a decomposition of X_i wrt. the norm equality, i.e., $h^*(X_i) = X_1^{|X_i|}$.

Initially, $\mathtt{A} = \{X_1\}$ and $\mathtt{S} = \{Y_\beta\}_\beta \cup \{X_2 \dots X_n\}$.

Invariant.

In the algorithm, the acyclic morphism h is determined by the leftmost prime factors and by the distinguished norm-reducing rules (5). Recall that $X_{\mathbf{lpf}(i)}$ is the leftmost prime factor of X_i wrt. $=_h$. The following invariant will be respected by h after each iteration of the REPEAT loop (note that (6) is a special case when $\mathbf{lpf}(i) = 1$):

$$h(X_i) = X_{\mathbf{lpf}(i)}\, \mathrm{suffix}_{|X_i|-|X_{\mathbf{lpf}(i)}|}(Y_{\alpha_i}) \quad \text{for every} \quad X_i \notin \mathtt{A}. \tag{7}$$

Why is (7) correct? It follows from the claim below.

▶ **Claim 1.** *Let h be an acyclic morphism such that the congruence $=_h$ is a norm-reducing bisimulation. If $X_i =_h X_j\delta$ then $h^*(\delta)$ is a suffix of $h^*(\alpha_i)$.*

Indeed: consider a norm-reducing transition $X_i \xrightarrow{a_i} \alpha_i$ and a matching transition of $X_j\delta$, say $X_j\delta \xrightarrow{a_i} \gamma\delta$; as X_j is an active variable, the process δ stays unchanged, and thus $\alpha_i =_h \gamma\delta$ as required.

From h to h'.

As the refinement of $=_h$ is included in $=_h$, a decomposition wrt. the refinement of $=_h$ is automatically a decomposition wrt. $=_h$. Thus a variable prime wrt. $=_h$ is still prime wrt.$=_{h'}$: $\mathtt{A} \subseteq \mathtt{A}'$. We do not introduce any new symbols for h': $\mathtt{A} \cup \mathtt{S} = \mathtt{A}' \cup \mathtt{S}'$, and hence $\mathtt{S} \supseteq \mathtt{S}'$. Thus $\mathtt{S}' = \{Y_\beta\}_\beta \cup \{X_2 \ldots X_n\} \setminus \mathtt{A}'$.

We assume that the occurrence table for h is available prior to each succesive iteration of the REPEAT loop (it was constructed in the previous iteration). As a side-effect of the consecutive invocations of **lpfactor** during one iteration of the REPEAT loop, the algorithm computes incrementally the occurrence table for h'. We assume that prior to each invocation of **lpfactor**(X_i, X_j) the occurrence table for h' is already computed for variables $\{X_1 \ldots X_{i-1}\}$.

Implementation of lpfactor(X_i, X_j).

The subroutine **lpfactor** is given, as its input, two variables X_i, X_j. Its task is to check whether X_j is the leftmost prime factor of X_i wrt. $\sim_{\text{n-r}}^{=_h \cap \exp(=_h)}$, the refinement of $=_h$.

It is assumed that prior to the call of **lpfactor**(X_i, X_j), all variables $X_2, \ldots, X_{i-1}$ have been processed by the outer FOR loop and either qualified to $\mathtt{A}'$, or have already a definition in h'. We thus assume that on $\{X_1 \ldots X_{i-1}\}$ the congruence $=_{h'}$ represented by h' agrees with $\sim_{\text{n-r}}^{=_h \cap \exp(=_h)}$. In particular $(h')^*(\alpha_i)$ is already defined, as all variables appearing in α_i are in $\{X_1 \ldots X_{i-1}\}$. It is also assumed that $X_i \notin \mathtt{A}'$ and $X_j \in \mathtt{A}'$. Recalling (7), the aim of **lpfactor**(X_i, X_j) is to check whether substituting **lpf**$(i) = j$ in (7) is correct for the refinement of h. I.e., the aim is to check the 'candidate':

$$h'(X_i) = X_j \,\text{suffix}_{|X_i|-|X_j|}(Y_{\alpha_i}). \tag{8}$$

The right-hand side of (8) is meaningful only when $(h')^*(Y_{\alpha_i}) = (h')^*(\alpha_i)$ has a suffix of norm $|X_i| - |X_j|$, say $\bar{\alpha} \in (\mathtt{A}')^*$; this is verified in point 1 below. The 'candidate' (8) is acceptable if the following condition holds:

$$(X_i,\ X_j\,\bar{\alpha}) \quad \in \quad \sim_{\text{n-r}}^{=_h \cap \exp(=_h)}, \tag{9}$$

By referring directly to Prop. 2, one sees that (9) is equivalent to

$$X_i =_h X_j\bar{\alpha} \ \text{ and } \ (X_i, X_j\,\bar{\alpha}) \in \exp(=_h) \ \text{ and } \ (X_i, X_j\,\bar{\alpha}) \in \text{n-r-exp}(=_{h'}).$$

(In the last condition we use the assumption that $=_{h'}$ agrees with $\sim_{\text{n-r}}^{=_h \cap \exp(=_h)}$ on variables from $\{X_1 \ldots X_{i-1}\}$.) These three conditions are verified in points 2, 3 and 4 below.

subroutine **lpfactor**$(X_i,\ X_j)$*:*

1. Check if $(h')^*(Y'_{\alpha_i})$ has a suffix of norm $l := |X_i| - |X_j|$. If no, return **false**. As $\mathtt{A} \subseteq \mathtt{A}'$ we conclude that $h^*(Y_{\alpha_i})$ has a suffix of norm l too, the fact to be needed in the following points.

2. Test if

$$X_i \ =_h \ X_j \operatorname{suffix}_l(Y_{\alpha_i}) \qquad (\text{in } \mathtt{A}^*). \tag{10}$$

   If this is not the case, return **false**.

3. For each $a \in \Sigma$, let $C_a := \{\alpha : X_i \xrightarrow{a} \alpha\}$ and $D_a := \{\beta : X_j \xrightarrow{a} \beta\}$; then for all $\alpha \in C_a$ and $\beta \in D_a$, test if

$$Y_\alpha \ =_h \ Y_\beta \operatorname{suffix}_l(Y_{\alpha_i}) \qquad (\text{in } \mathtt{A}^*). \tag{11}$$

   If the bisimulation expansion condition (for each $\alpha \in C_a$, there exists $\beta \in D_a$ such that (11) holds; and for each $\beta \in D_a$, there exists $\alpha \in C_a$ such that (11) holds) is not satisfied return **false**.

4. For each $a \in \Sigma$, let $C_a := \{\alpha : X_i \xrightarrow{a}_{\text{n-r}} \alpha\}$ and $D_a := \{\beta : X_j \xrightarrow{a}_{\text{n-r}} \beta\}$; then for all $\alpha \in C_a$ and $\beta \in D_a$, test if

$$Y_\alpha \ =_{h'} \ Y_\beta \operatorname{suffix}_l(Y_{\alpha_i}) \qquad (\text{in } (\mathtt{A}')^*). \tag{12}$$

   If the bisimulation expansion condition, with (12) in place of (11), is not satisfied return **false**.

5. Extend h' by $h'(X_i) = X_j \operatorname{suffix}_{|X_i|-|X_j|}(Y_{\alpha_i})$ and return **true**.

In case when the 'candidate' (9) is checked succesfully, as a side-effect of the invocation of **lpfactor** the acyclic morphism h' is extended in point 5.

It remains now to explain how the equality tests (10), (11) and (12) are implemented. Basing on the the same insight as in Theorem 7 we prove:

▶ **Lemma 10.** *Each of equality tests* (10), (11) *may be solved in time* $\mathcal{O}(N)$*. All equality tests* (12) *during one iteration of the* REPEAT *loop require* $\mathcal{O}(N^4)$ *time.*

Total running time.

The time needed for a single iteration of the REPEAT loop is devoted to two tasks: (I) construction of the occurrence table for h', as a side-effect of solving tests (12), and (II) solving the equality tests (10) and (11). Task (I) requires $\mathcal{O}(N^4)$ total time, by Lemma 10 and Theorem 7, knowing that the depth of h is at most n (note that the depth is so even after increasing by a logarithmic factor when transforming h to the binary form). Concerning (II), there are $\mathcal{O}(N^2)$ equality tests in a single iteration of the REPEAT loop and each of them requires $\mathcal{O}(N)$ time by Lemma 10, hence the latter task is time-dominated by the former one. As the number of iterations is at most n, we get total time $\mathcal{O}(N^5)$, as stated in Theorem 2.

Example.

As an example, we analyze a run of our algorithm for the following input process definition:

$$X \xrightarrow{a} \epsilon \qquad Y \xrightarrow{a} \epsilon \qquad Y \xrightarrow{b} Y \qquad Y \xrightarrow{b} X \qquad Z \xrightarrow{a} X$$
$$Z \xrightarrow{b} Z \qquad Z \xrightarrow{b} YY \qquad T \xrightarrow{a} YY \qquad W \xrightarrow{a} ZZ \qquad W \xrightarrow{b} W$$

Variables are ordered as follows: $X < Y < Z < T < W$. The compression issue, i.e., representation of the approximating congruences by acyclic morphisms, are completely omitted here for simplicity. We fix:

$$\alpha_X = \epsilon \qquad 1\alpha_Y = \epsilon \qquad \alpha_Z = X \qquad \alpha_T = YY \qquad \alpha_W = ZZ.$$

The initial decomposition is: $Y \equiv X$, $Z \equiv XX$, $T \equiv XXX$, $W \equiv XXXXX$; variable X is the only prime.

Let us analyze the first iteration of the REPEAT loop; the refinement of $\equiv$ computed in this iteration we denote by $\equiv'$. We check that **lpfactor**(Y, X) yields **false**, because $(Y, X) \notin \exp(\equiv)$: Y has a b-move, X has not. So Y becomes prime. Next we check that **lpfactor**(Z, X) yields **false** because $(Z, XX = X\alpha_Z) \notin \exp(\equiv)$. We check that **lpfactor**(Z, Y) yields **true**, so we put $Z \equiv' YX = Y\alpha_Z$. Next we check that **lpfactor**(T, X) yields **true**, hence $T \equiv' XYY = X\alpha_T$. Now we proceed with processing of the last variable W. We check that **lpfactor**(W, X) yields **false**, because $(W, XYXYX \equiv' X\alpha_W) \notin \exp(\equiv)$: W has a b-move, X has not. Finally, we check that **lpfactor**(W, Y) yields **true**, so now $W \equiv' Y\alpha_W \equiv' YZZ \equiv' YYXYX$.

After the first iteration, the decomposition is: $Z \equiv YX$, $T \equiv XYY$, $W \equiv YYXYX$; the primes are $\mathtt{A} = \{X, Y\}$.

Now we proceed with the analysis of the second iteration of the REPEAT loop. We check that **lpfactor**(Z, Y) yields **false**, because $(Z, YX) \notin \exp(\equiv)$: there is no good Duplicator's response to the move $Z \xrightarrow{b} YY$. In consequence, Z becomes prime. Next we check that **lpfactor**(T, X) yields **true**. In consequence, $T \equiv' XYY$. Finally, we process variable W. We check that **lpfactor**(W, Y) yields **false**, as $(W, YZZ) \notin \exp(\equiv)$: there is no good response to the move $Y \xrightarrow{b} X$. The next candidate for the left-most prime factor is Z, a 'fresh' prime. We check that **lpfactor**(W, Z) yields **false**, but the reason is different than before: $\alpha_W = ZZ$ has no suffix of norm $|W| - |Z| = 3$.

After the second iteration, we have four primes $\mathtt{A} = \{X, Y, Z, W\}$ and the decomposition is $T \equiv XYY$.

The third iteration is the last one as A does not change any more.

Acknowledgements.

We are very grateful to Wojtek Rytter and Sibylle Fröschle for many helpful discussions.

References

1 Y. Bar-Hillel, M. Perles, and S. Shamir. On formal properties of simple phrase structure grammars. *Zeitschrift fuer Phonetik, Sprachwissenschaft, und Kommunikationsforschung*, 14:143–177, 1961.

2 C. Bastien, J. Czyżowicz, W. Fraczak, and W. Rytter. Prime normal form and equivalence of simple grammars. In *Proc. CIAA'05*, volume 3845 of *LNCS*, pages 79–90. Springer-Verlag, 2005.

3 J. Beaten, J. Bergstra, and J. Klop. Decidability of bisimulation equivalence for processes generating context-free languages. In *Proc. PARLE'87*, volume 259 of *LNCS*, pages 94–113. Springer-Verlag, 1987.

4 D. Caucal. Graphes canoniques des graphes algébraiques. *Informatique Théoretique et Applications (RAIRO)*, 24(4):339–352, 1990.

5 S. Christensen, Y. Hirshfeld, and C. Stirling. Bisimulation equivalence is decidable for all context-free processes. *Information and Computation*, 12(2):143–148, 1995.

6 W. Czerwiński, S. Fröschle, and S. Lasota. Partially-commutative context-free processes. In *Proc. CONCUR'09*, volume 5710 of *LNCS*, pages 259–273. Springer-Verlag, 2009.

7 E.P. Friedman. The inclusion problem for simple languages. *Theoretical Computer Science*, 1:297–316, 1976.

8 R. v. Glabbeek. The linear time - branching time spectrum. In *Proc. CONCUR'90*, pages 278–297, 1990.

9 Y. Hirshfeld, M. Jerrum, and F. Moller. A polynomial algorithm for deciding bisimilarity on normed context-free processes. *Theoretical Computer Science*, 15:143–159, 1996.

10 H. Huettel and C. Stirling. Actions speak louder than words: proving bisimilarity for context-free processes. In *Proc. LICS'91*, pages 376–386. IEEE Computer Society Press, 1991.

11 D. Huynh and L. Tian. Deciding bisimilarity of normed context-free processes is in Σ_2^P. *Theoretical Computer Science*, 123:183–197, 1994.

12 A. Korenjak and J. Hopcroft. Simple deterministic languages. In *Proc. 7th Annual IEEE Symposium on Switching and Automata Theory*, pages 36–46, 1966.

13 S. Lasota and W. Rytter. Faster algorithm for bisimulation equivalence of normed context-free processes. In *Proc. MFCS'06*, volume 4162 of *LNCS*, pages 646–657. Springer-Verlag, 2006.

14 Y. Lifshits. Solving classical string problems an compressed texts. In *Combinatorial and Algorithmic Foundations of Pattern and Association Discovery*, 2006.

15 A. Shinohara, M. Miyazaki, and M. Takeda. An improved pattern-matching algorithm for strings in terms of straight-line programs. *Journal of Discrete Algorithms*, 1(1):187–204, 2000.

Generalizing the powerset construction, coalgebraically *

Alexandra Silva[1], Filippo Bonchi[2], Marcello M. Bonsangue[1,3], and Jan J. M. M. Rutten[1,4]

1 **Centrum Wiskunde & Informatica**
2 **CNRS - LIP, ENS Lyon**
3 **LIACS - Leiden University**
4 **Radboud University Nijmegen**

Abstract

Coalgebra is an abstract framework for the uniform study of different kinds of dynamical systems. An endofunctor F determines both the type of systems (F-coalgebras) and a notion of behavioral equivalence ($\sim_F$) amongst them. Many types of transition systems and their equivalences can be captured by a functor F. For example, for deterministic automata the derived equivalence is language equivalence, while for non-deterministic automata it is ordinary bisimilarity. The powerset construction is a standard method for converting a nondeterministic automaton into an equivalent deterministic one as far as language is concerned. In this paper, we lift the powerset construction on automata to the more general framework of coalgebras with structured state spaces. Examples of applications include partial Mealy machines, (structured) Moore automata, and Rabin probabilistic automata.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.272

1 Introduction

Coalgebra is by now a well established general framework for the study of the behaviour of large classes of dynamical systems, including various kinds of automata (deterministic, probabilistic etc.) and infinite data types (streams, trees and the like). For a functor $F\colon \mathbf{Set} \to \mathbf{Set}$, an F-coalgebra is a pair (X, f), consisting of a set X of states and a function $f\colon X \to F(X)$ defining the observations and transitions of the states. Coalgebras generally come equipped with a standard notion of equivalence called *F-behavioural equivalence* that is fully determined by their (functor) type F. Moreover, for most functors F there exists a *final* coalgebra into which any F-coalgebra is mapped by a unique homomorphism that identifies all F-equivalent states.

Much of the coalgebraic approach can be nicely illustrated with deterministic automata (DA), which are coalgebras of the functor $D(X) = 2 \times X^A$. In a DA, two states are D-equivalent precisely when they accept the same language. The set 2^{A^*} of all formal languages constitutes a final D-coalgebra, into which every DA is mapped by a homomorphism that sends any state to the language it accepts.

It is well-known that *non-deterministic* automata (NDA) often provide more efficient (smaller) representations of formal languages than DA's. Language acceptance of NDA's is typically defined by turning them into DA's via the *powerset construction*. Coalgebraically this works as follows. NDA's are coalgebras of the functor $N(X) = 2 \times \mathcal{P}_\omega(X)^A$, where

* This work was carried out during the second author's tenure of an ERCIM "Alain Bensoussan" Fellowship Programme. The fourth author is partially supported by the Fundação para a Ciência e a Tecnologia, Portugal, under grant number SFRH/BD/27482/2006

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 272–283

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

$\mathcal{P}_\omega$ is the finite powerset. An N-coalgebra $(X, f\colon X \to 2 \times \mathcal{P}_\omega(X)^A)$ is *determinized* by transforming it into a D-coalgebra $(\mathcal{P}_\omega(X), f^\sharp\colon \mathcal{P}_\omega(X) \to 2 \times \mathcal{P}_\omega(X)^A)$ (for details see Section 3). Then, the language accepted by a state s in the NDA (X, f) is defined as the language accepted by the state $\{s\}$ in the DA $(\mathcal{P}_\omega(X), f^\sharp)$.

For a second variation on DA's, we look at *partial automata* (PA): coalgebras of the functor $P(X) = 2 \times (1 + X)^A$, where for certain input letters transitions may be undefined. Again, one is often interested in the DA-behaviour (i.e., language acceptance) of PA's. This can be obtained by turning them into DA's using *totalization*. Coalgebraically, this amounts to the transformation of a P-coalgebra $(X, f\colon X \to 2 \times (1 + X)^A)$ into a D-coalgebra $(1 + X, f^\sharp\colon 1 + X \to 2 \times (1 + X)^A)$.

Although the two examples above may seem very different, they are both instances of one and the same phenomenon, which it is the goal of the present paper to describe at a general level. Both with NDA's and PA's, two things happen at the same time: (i) more (or, more generally, different types of) transitions are allowed, as a consequence of changing the functor type by replacing X by $\mathcal{P}_\omega(X)$ and $(1 + X)$, respectively; and (ii) the behaviour of NDA's and PA's is still given in terms of the behaviour of the original DA's (language acceptance).

For a large family of F-coalgebras, both (i) and (ii) can be captured simultaneously with the help of the categorical notion of *monad*, which generalizes the notion of algebraic theory. The structuring of the state space X can be expressed as a change of functor type from $F(X)$ to $F(T(X))$. In our examples above, both the functors $T_1(X) = \mathcal{P}_\omega(X)$ and $T_2(X) = 1 + X$ are monads, and NDA's and PA's are obtained from DA's by changing the original functor type $D(X)$ into $N(X) = D(T_1(X))$ and $P(X) = D(T_2(X))$. Regarding (ii), one assigns F-semantics to an FT-coalgebra (X, f) by transforming it into an F-coalgebra $(T(X), f^\sharp)$, again using the monad T. In our examples above, the determinization of NDA's and the totalization of PA's consists of the transformation of N- and P-coalgebras (X, f) into D-coalgebras $(T_1(X), f^\sharp)$ and $(T_2(X), f^\sharp)$, respectively.

We shall investigate general conditions on the functor types under which the above constructions can be applied: for one thing, one has to ensure that the FT-coalgebra map f induces a suitable F-coalgebra map $f^\sharp$. Our results will lead to a uniform treatment of all kinds of existing and new variations of automata, that is, FT-coalgebras, by an algebraic structuring of their state space through a monad T. Furthermore, we shall prove a number of general properties that hold in all situations similar to the ones above. For instance, there is the notion of N-behavioural equivalence with which NDA's, being N-coalgebras, come equipped. It coincides with the well-known notion of Park-Milner bisimilarity from process algebra. A general observation is that if two states in an NDA are N-equivalent then they are also D- (that is, language-) equivalent. For PA's, a similar statement holds. One further contribution of this paper is a proof of these statements, once and for all for all FT-coalgebras under consideration.

Coalgebras of type FT were studied in [15, 2, 11]. In [2, 11] the main concern was definitions by coinduction, whereas in [15] a proof principle was also presented. All in all, the present paper can be seen as the understanding of the aforementioned papers from a new perspective, presenting a uniform view on various automata constructions and equivalences.

The structure of the paper is as follows. After preliminaries (Section 2) and the details of the motivating examples above (Section 3), Section 4 presents the general construction as well as many more examples. In Section 5, a large family of automata (technically: functors) is characterized to which the constructions above can be applied. Section 6 discusses related work and presents pointers to future work. A technical report [27] contains all the proofs as well as further examples.

2 Background

In this section we introduce the preliminaries on coalgebras and algebras. First, we fix some notation on sets. We will denote sets by capital letters $X, Y, \ldots$ and functions by lower case letters $f, g, \ldots$ Given sets X and Y, $X \times Y$ is the cartesian product of X and Y (with the usual projection maps π_1 and π_2), $X + Y$ is the disjoint union (with injection maps κ_1 and κ_2) and X^Y is the set of functions $f{:}Y \to X$. The collection of finite subsets of X is denoted by $\mathcal{P}_\omega(X)$, while the collection of full-probability distributions with finite support is $\mathcal{D}_\omega(X) = \{f\colon X \to [0,1] \mid f \text{ finite support and } \sum_{x\in X} f(x) = 1\}$. For a set of letters A, A^* denotes the set of all words over A; ϵ the empty word; and $w_1 \cdot w_2$ (and $w_1 w_2$) the concatenation of words $w_1, w_2 \in A^*$.

2.1 Coalgebras

A coalgebra is a pair $(X, f\colon X \to F(X))$, where X is a set of states and $F\colon \mathbf{Set} \to \mathbf{Set}$ is a functor. The functor F, together with the function f, determines the *transition structure* (or dynamics) of the F-coalgebra [22].

An *F-homomorphism* from an F-coalgebra (X, f) to an F-coalgebra (Y, g) is a function $h\colon X \to Y$ preserving the transition structure, *i.e.*, $g \circ h = F(h) \circ f$.

An F-coalgebra (Ω, ω) is said to be *final* if for any F-coalgebra (X, f) there exists a unique F-homomorphism $[\![-]\!]_X\colon X \to \Omega$. All the functors considered in examples in this paper have a final coalgebra.

Let (X, f) and (Y, g) be two F-coalgebras. We say that the states $x \in X$ and $y \in Y$ are *behaviourally equivalent*, written $x \sim_F y$, if and only if they are mapped into the same element in the final coalgebra, that is $[\![x]\!]_X = [\![y]\!]_Y$.

2.2 Algebras

Monads can be thought of as a generalization of algebraic theories. A *monad* $\mathbf{T} = (T, \mu, \eta)$ is a triple consisting of an endofunctor T on **Set** and two natural transformations: a *unit* $\eta\colon Id \Rightarrow T$ mapping a set X to its free algebra $T(X)$, and a *multiplication* $\mu\colon T^2 \Rightarrow T$. They satisfy the following commutative laws

$$\mu \circ \eta_T = id_T = \mu \circ T\eta \quad \text{and} \quad \mu \circ \mu_T = \mu \circ T\mu.$$

Sometimes it is more convenient to represent a monad $\mathbf{T}$, equivalently, as a *Kleisli triple* $(T, (\_)^\sharp, \eta)$ [17], where T assigns a set $T(X)$ to each set X, the unit η assigns a function $\eta_X\colon X \to T(X)$ to each set X, and the extension operation $(\_)^\sharp$ assigns to each $f\colon X \to T(Y)$ a function $f^\sharp\colon T(X) \to T(Y)$, such that,

$$f^\sharp \circ \eta_X = f \qquad (\eta_X)^\sharp = id_{T(X)} \qquad (g^\sharp \circ f)^\sharp = g^\sharp \circ f^\sharp\,,$$

for $g\colon Y \to T(Z)$. Monads are frequently referred to as *computational types* [18]. We list now a few examples. In what follows, $f\colon X \to T(Y)$ and $c \in T(X)$.

Nondeterminism $T(X) = \mathcal{P}_\omega(X)$; η_X is the singleton map $x \mapsto \{x\}$; $f^\sharp(c) = \bigcup_{x\in c} f(x)$.

Partiality $T(X) = 1 + X$ where $1 = \{*\}$ represents a terminating (or diverging) computation; η_X is the injection map $\kappa_2\colon X \to 1 + X$; $f^\sharp(\kappa_1(*)) = \kappa_1(*)$ and $f^\sharp(\kappa_2(x)) = f(x)$.

Further examples of monads include: exceptions ($T(X) = E + X$), side-effects ($T(X) = (S \times X)^S$), interactive output ($T(X) = \mu v.X + (O \times v) \cong O^* \times X$) and full-probability

$(T(X) = \mathcal{D}_\omega(X))$. We will use all these monads in our examples and we will define η_X and $f^\sharp$ for each later in Section 4.1.

A $\mathbf{T}$-*algebra* of a monad $\mathbf{T}$ is a pair (X, h) consisting of a set X, called carrier, and a function $h\colon T(X) \to X$ such that $h \circ \mu_X = h \circ Th$ and $h \circ \eta_X = id_X$. A T-homomorphism between two $\mathbf{T}$-algebras (X, h) and (Y, k) is a function $f\colon X \to Y$ such that $f \circ h = k \circ Tf$. $\mathbf{T}$-algebras and their homomorphisms form the so-called *Eilenberg-Moore category* $\mathbf{Set}^{\mathbf{T}}$. There is a forgetful functor $U^{\mathbf{T}}\colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ defined by

$$U^{\mathbf{T}}((X, h)) = X \quad \text{and} \quad U^{\mathbf{T}}(f\colon (X, h) \to (Y, k)) = f\colon X \to Y\,.$$

The forgetful functor $U^{\mathbf{T}}$ has left adjoint $X \mapsto (T(X), \mu_X\colon TT(X) \to T(X))$, mapping a set X to its free $\mathbf{T}$-algebra. If $f\colon X \to Y$ with (Y, h) a $\mathbf{T}$-algebra, the unique $\mathbf{T}$-homomorphism $f^\sharp\colon (T(X), \mu_X) \to (Y, h)$ with $f^\sharp \circ \eta_X = f$ is given by

$$f^\sharp\colon T(X) \xrightarrow{Tf} T(Y) \xrightarrow{h} Y\,.$$

The function $f^\sharp\colon (T(X), \mu_X) \to (T(Y), \mu_Y)$ coincides with function extension for a Kleisli triple. For the monad $\mathcal{P}_\omega$ the associated Eilenberg-Moore category is the category of join semi-lattices, whereas for the monad $1 + -$ is the category of pointed sets.

3 Motivating examples

In this section, we introduce two motivating examples. We will present two constructions, the determinization of a non-deterministic automaton and the totalization of a partial automaton, which we will later show to be an instance of the same, more general, construction.

3.1 Non-deterministic automata

A deterministic automaton (DA) over the input alphabet A is a pair $(X, \langle o, t\rangle)$, where X is a set of states and $\langle o, t\rangle\colon X \to 2 \times X^A$ is a function with two components: o, the output function, determines if a state x is final ($o(x) = 1$) or not ($o(x) = 0$); and t, the transition function, returns for each input letter a the next state. DA's are coalgebras for the functor $2 \times Id^A$. The final coalgebra of this functor is $(2^{A^*}, \langle \epsilon, (-)_a\rangle)$ where 2^{A^*} is the set of languages over A and $\langle \epsilon, (-)_a\rangle$, given a language L, determines whether or not the empty word is in the language ($\epsilon(L) = 1$ or $\epsilon(L) = 0$, resp.) and, for each input letter a, returns the *derivative* of L: $L_a = \{w \in A^* \mid aw \in L\}$. From any DA, there is a unique map l into 2^{A^*} which assigns to each state its behaviour (that is, the language that the state recognizes).

$$\begin{array}{ccc} X & \xrightarrow{\quad l \quad} & 2^{A^*} \\ {\scriptstyle \langle o,t\rangle}\downarrow & & \downarrow{\scriptstyle \langle \epsilon,(-)_a\rangle} \\ 2 \times X^A & \xrightarrow[id \times l^A]{} & 2 \times (2^{A^*})^A \end{array}$$

A non-deterministic automaton (NDA) is similar to a DA but the transition function gives a set of next-states for each input letter instead of a single state. Thus, an NDA over the input alphabet A is a pair $(X, \langle o, \delta\rangle)$, where X is a set of states and $\langle o, \delta\rangle\colon X \to 2 \times (\mathcal{P}_\omega(X))^A$ is a pair of functions with o as before and where δ determines for each input letter a a set of possible next states. In order to compute the language recognized by a state x of an NDA $\mathcal{A}$, it is usual to first determinize it, constructing a DA $\mathbf{det}(\mathcal{A})$ where the state space is $\mathcal{P}_\omega(X)$,

and then compute the language recognized by the state $\{x\}$ of $\mathbf{det}(\mathcal{A})$. Next, we describe in coalgebraic terms how to construct the automaton $\mathbf{det}(\mathcal{A})$.

Given an NDA $\mathcal{A} = (X, \langle o, \delta\rangle)$, we construct $\mathbf{det}(\mathcal{A}) = (\mathcal{P}_\omega(X), \langle \overline{o}, t\rangle)$, where, for all $Y \in \mathcal{P}_\omega(X)$, $a \in A$, the functions $\overline{o}\colon \mathcal{P}_\omega(X) \to 2$ and $t\colon \mathcal{P}_\omega(X) \to \mathcal{P}_\omega(X)^A$ are

$$\overline{o}(Y) = \begin{cases} 1 & \exists_{y\in Y} o(y) = 1 \\ 0 & \text{otherwise} \end{cases} \qquad t(Y)(a) = \bigcup_{y\in Y} \delta(y)(a).$$

The automaton $\mathbf{det}(\mathcal{A})$ is such that the language $l(\{x\})$ recognized by $\{x\}$ is the same as the one recognized by x in the original NDA $\mathcal{A}$ (more generally, the language recognized by state X of $\mathbf{det}(\mathcal{A})$ is the union of the languages recognized by each state x of $\mathcal{A}$).

We summarize the situation above with the following commuting diagram:

$$\begin{array}{ccc} X & \xrightarrow{\{\cdot\}} & \mathcal{P}_\omega(X) \xrightarrow{l} 2^{A^*} \\ {\scriptstyle\langle o,\delta\rangle}\downarrow & \swarrow_{\langle \overline{o}, t\rangle} & \downarrow{\scriptstyle\langle \epsilon, (-)_a\rangle} \\ 2 \times \mathcal{P}_\omega(X)^A & \xrightarrow[id\times l^A]{} & 2 \times (2^{A^*})^A \end{array}$$

We note that the language semantics of NDA's, presented in the above diagram, can also be obtained as an instance of the abstract definition scheme of λ-coinduction [2, 11].

3.2 Partial automata

A partial automaton (PA) over the input alphabet A is a pair $(X, \langle o, \partial\rangle)$ consisting of a set of states X and a pair of functions $\langle o, \partial\rangle\colon X \to 2 \times (1 + X)^A$, with $o\colon X \to 2$ as for DA and $\partial\colon X \to (1 + X)^A$ a transition function, which for any input letter a is either undefined (no a-labelled transition takes place) or specifies the next state that is reached. PA's are coalgebras for the functor $2 \times (1 + Id)^A$. Given a PA $\mathcal{A}$, we can construct a total (deterministic) automaton $\mathbf{tot}(\mathcal{A})$ by adding an extra *sink* state to the state space: every undefined a-transition from a state x is then replaced by a a-labelled transition from x to the sink state. More precisely, given a PA $\mathcal{A} = (X, \langle o, \partial\rangle)$, we construct $\mathbf{tot}(\mathcal{A}) = (1 + X, \langle \overline{o}, t\rangle)$, where

$$\begin{array}{ll} \overline{o}(\kappa_1(*)) = 0 & t(\kappa_1(*))(a) = \kappa_1(*) \\ \overline{o}(\kappa_2(x)) = o(x) & t(\kappa_2(x))(a) = \partial(x)(a) \end{array}$$

The language $l(x)$ recognized by a state x will be precisely the language recognized by x in the original partial automaton. Moreover, the new sink state recognizes the empty language. Again we summarize the situation above with the help of following commuting diagram, which illustrates the similarities between both constructions:

$$\begin{array}{ccc} X & \xrightarrow{\kappa_2} & 1 + X \xrightarrow{l} 2^{A^*} \\ {\scriptstyle\langle o,\partial\rangle}\downarrow & \swarrow_{\langle \overline{o}, t\rangle} & \downarrow{\scriptstyle\langle \epsilon, (-)_a\rangle} \\ 2 \times (1 + X)^A & \xrightarrow[id\times l^A]{} & 2 \times (2^{A^*})^A \end{array}$$

4 Algebraically structured coalgebras

In this section we present a general framework where both motivating examples can be embedded and uniformly studied. We will consider coalgebras for which the functor type FT

can be decomposed into a transition type F specifying the relevant dynamics of a system and a monad T providing the state space with an algebraic structure. For simplicity, we fix our base category to be **Set**.

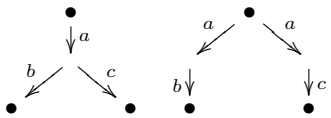
We will study coalgebras $f\colon X \to FT(X)$ for a functor F and a monad $\mathbf{T}$ such that $FT(X)$ is a $\mathbf{T}$-algebra, that is $FT(X)$ is the carrier of a $\mathbf{T}$-algebra $(FT(X), h)$. In the motivating examples, F would be instantiated to $2 \times Id^A$ (in both) and T to $\mathcal{P}_\omega$, for NDAs, and to $1 + -$ for PAs. The condition that $FT(X)$ is a $\mathbf{T}$-algebra would amount to require that $2 \times \mathcal{P}_\omega(X)^A$ is a join-semilattice, for NDAs, and that $2 \times (1 + X)^A$ is a pointed set, for PAs. This is indeed the case, since the set 2 can be regarded both as a join-semilattice ($2 \cong \mathcal{P}_\omega(1)$) or as a pointed set ($2 \cong 1+1$) and, moreover, products and exponentials preserve the algebra structure.

The inter-play between the transition type F and the computational type $\mathbf{T}$ (more precisely, the fact that $FT(X)$ is a $\mathbf{T}$-algebra) will allow each coalgebra $f\colon X \to FT(X)$ to be extended uniquely to a T-algebra morphism $f^\sharp\colon (T(X), \mu_X) \to (FT(X), h)$ which makes the following diagram commute.

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & T(X) \\ {\scriptstyle f}\downarrow & \swarrow_{f^\sharp} & \\ FT(X) & & \end{array} \qquad f^\sharp \circ \eta_X = f$$

Intuitively, $\eta_X\colon X \to T(X)$ is the inclusion of the state space of the coalgebra $f\colon X \to FT(X)$ into the structured state space $T(X)$, and $f^\sharp\colon T(X) \to FT(X)$ is the extension of the coalgebra f to $T(X)$.

Next, we will study the behaviour of a given state or, more generally, we would like to say when two states x_1 and x_2 are equivalent. The obvious choice for an equivalence would be FT-behavioural equivalence. However, this equivalence is not exactly what we are looking for. In the motivating example of non-deterministic automata we wanted two states to be equivalent if they recognize the same language. If we would take the equivalence arising from the functor $2 \times \mathcal{P}_\omega(Id)^A$ we would be distinguishing states that recognize the same language but have difference branching types, as in the following example.



We now define a new equivalence, which will *absorb* the effect of the monad T.

We say that two elements x_1 and x_2 in X are *F-equivalent with respect to a monad* $\mathbf{T}$, written $x_1 \approx_F^T x_2$, if and only if $\eta_X(x_1) \sim_F \eta_X(x_2)$. The equivalence $\sim_F$ is just F-behavioural equivalence for the F-coalgebra $f^\sharp\colon T(X) \to FT(X)$.

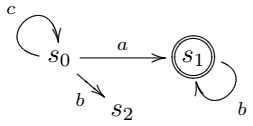
If the functor F has a final coalgebra (Ω, ω) , we can capture the semantic equivalence above in the following commuting diagram

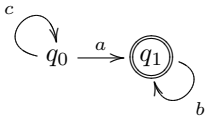
$$\begin{array}{ccccc} X & \xrightarrow{\eta_X} & T(X) & \xrightarrow{[\![-]\!]} & \Omega \\ {\scriptstyle f}\downarrow & \swarrow_{f^\sharp} & & & \downarrow{\scriptstyle \omega} \\ FT(X) & & \xrightarrow{F[\![-]\!]} & & F(\Omega) \end{array} \tag{1}$$

Back to our first example, two states x_1 and x_2 of an NDA (in which T is instantiated to $\mathcal{P}_\omega$ and F to $2 \times Id^A$) would satisfy $x_1 \approx_F^T x_2$ if and only if they recognize the same language (recall that the final coalgebra of the functor $2 \times Id^A$ is 2^{A^*}).

It is also interesting to remark the difference between the two equivalences in the case of partial automata. The coalgebraic semantics of PAs [24] is given in terms of pairs of prefix-closed languages $\langle V, W\rangle$ where V contains the words that are accepted (that is, are the label of a path leading to a final state) and W contains all words that label any path (that is all that are in V plus the words labeling paths leading to non-final states). We exemplify what V and W would be in the following examples for state s_0 and q_0.

$W = c^* + c^*b + c^*ab^*$
$V = c^*ab^*$





$W = c^* + c^*ab^*$
$V = c^*ab^*$

Thus, state s_0 and q_0 would be distinguished by FT-equivalence (for $F = 2 \times Id^A$ and $T = 1 + -$) but they are equivalent with respect to the monad $1 + -$, $s_0 \approx_F^T q_0$, since they accept the same language.

We will show in Section 5 that the equivalence $\sim_{FT}$ is contained in $\approx_F^T$.

4.1 Examples

In this section we show more examples of applications of the framework above.

4.1.1 Partial Mealy machines

A partial Mealy machine is a set of states X together with a function $t\colon X \to (B \times (1+X))^A$, where A is a set of inputs and B is a set of output values (with a distinguished value $\perp$). For each state s and for each input a the automaton produces an output value and either terminates or continues to a next state. Applying the framework above we will be *totalizing* the automaton, similarly to what happened in the example of partial automata, by adding an extra state to the state space which will act as a sink state. The behaviour of the totalized automaton is given by the set of causal functions from A^ω (infinite sequences of A) to B^ω, which we denote by $\Gamma(A^\omega, B^\omega)$ [23]. A function $f\colon A^\omega \to B^\omega$ is causal if, for $\sigma \in A^\omega$, the n-th value of the output stream $f(\sigma)$ depends only on the first n values of the input stream σ.

$$\begin{array}{ccc} X & \xrightarrow{\quad} & 1 + X \overset{[\![-]\!]}{- - - - - - - - - - \to} \Gamma(A^\omega, B^\omega) \\ {\scriptstyle t}\downarrow & \swarrow_{t^\sharp} & [\![\kappa_1(*)]\!](\sigma) = (\perp, \perp, \ldots) \\ & & [\![\kappa_2(s)]\!](a{:}\sigma) = b{:}([\![n]\!](\sigma)) \quad \downarrow \\ & & \text{where } \langle b, n\rangle = t(s)(a) \\ (B \times (1+X))^A & - - - - - - - - - - - - - - - - \to & (B \times \Gamma(A^\omega, B^\omega))^A \end{array}$$

4.1.2 Structured Moore automata

In the following examples we look at the functor $F(X) = T(B) \times X^A$, for B and A arbitrary sets and $\mathbf{T} = (T, \eta, (-)^\sharp)$ an arbitrary monad. This represents Moore automata with outputs in $T(B)$ and inputs in A. For any set X, $FT(X)$ has a $\mathbf{T}$-algebra lifting and the final coalgebra of F is $T(B)^{A^*}$. The final map $[\![-]\!]\colon T(X) \to T(B)^{A^*}$ is defined below.

$$\begin{array}{ccc} X & \xrightarrow{\eta_X} & T(X) \overset{[\![-]\!]}{- - - - - - - - - - \to} T(B)^{A^*} \\ {\scriptstyle \langle o,t\rangle}\downarrow & \swarrow_{\langle o,t\rangle^\sharp} & [\![m]\!](\epsilon) = o^\sharp(m) \quad \downarrow {\scriptstyle \langle \epsilon, (-)_a\rangle} \\ & & [\![m]\!](aw) = [\![t^\sharp(m)(a)]\!](w) \\ T(B) \times (T(X))^A & - - - - - - - - - - - - - - - - \to & T(B) \times (T(T(B)^{A^*}))^A \end{array}$$

4.1.2.1 Moore automata with exceptions

Consider $T(X) = E + X$, with E a set of exceptions, $\eta(x) = \kappa_2(x)$ and, for a function $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is defined as $f^\sharp = [id, f]$.

An FT-coalgebra $\langle o, t\rangle\colon X \to (E + B) \times (E + X)^A$ will associate with every state s an output value (either in B or an exception in E) and, for each input a, a next state or an exception. The behaviour of a state x, given by $[\![\eta(x)]\!]$, will be a formal power series over A with output values in $E + B$ (that is, a function from A^* to $E + B$), defined as follows:

$$[\![\kappa_1(e)]\!](w) = \kappa_1(e) \qquad [\![\kappa_2(s)]\!](\epsilon) = o(s) \qquad [\![\kappa_2(s)]\!](aw) = [\![t(s)(a)]\!](w).$$

4.1.2.2 Moore automata with side effects

Consider $T(X) = (S \times X)^S$, with S a set of side-effects, $\eta(x) = \lambda s.\langle s, x\rangle$ and, for a function $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is defined as $f^\sharp(g)(s) = f(x)(s')$ where $\langle s', x\rangle = g(s)$.

Take now an FT-coalgebra $\langle o, t\rangle\colon X \to (B \times S)^S \times ((S \times X)^S)^A$ and let us explain the intuition behind this automaton type. Let S be the set of side effects (for instance, one could take $S = V^L$, functions associating memory locations to values). The set $S \times X$ can be interpreted as the configurations of the automaton, where S contains information about the state of the system and X about the control of the system. Then, we can think of $o\colon X \to (S \times B)^S$ as a function that for each configuration $S \times X$ provides an output and the new state of the system (note that $X \to (S \times B)^S \cong S \times X \to S \times B$). The transition function $t\colon X \to ((S \times X)^S)^A$ gives a new configuration for each input letter and current configuration (again we use the fact that $X \to ((S \times X)^S)^A \cong S \times X \to (S \times X)^A$).

The behaviour of a state x will be given by $[\![\eta(x)]\!]$, defined below, and it will be a function that for each configuration and for each sequence of actions returns an output value and a side effect.

$$\begin{array}{lcl} [\![g]\!](\epsilon)(s) & = & o(x)(s') \text{ where } \langle s', x\rangle = g(s) \\ [\![g]\!](aw_1) & = & [\![\lambda s.t(s)(a)(s')]\!](w_1) \text{ where } \langle s', x\rangle = g(s) \end{array}$$

4.1.2.3 Moore automata with interactive output

Consider $T(X) = \mu v.X + (O \times v) \cong O^* \times X$, with O a set of outputs, $\eta(x) = \langle \epsilon, x\rangle$ and, for $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is given by $f^\sharp(\langle w, x\rangle) = \langle ww', x'\rangle$ where $\langle w', x'\rangle = f(x)$. Take an FT-coalgebra $\langle o, t\rangle\colon X \to (O^* \times B) \times (O^* \times X)^A$. For $B = 1$, this coincides with a *(total) subsequential transducer* [8]: $o\colon X \to O^*$ is the terminal output function; $t\colon X \to (O^* \times X)^A$ is the pairing of the output function and the next state-function.

The behaviour of a state x will be given by $[\![\eta(x)]\!] = [\![\langle \epsilon, x\rangle]\!]$, where, for every $\langle w, x\rangle \in O^* \times X$, $[\![\langle w, x\rangle]\!]\colon A^* \to B^*$, is given by

$$[\![\langle w, x\rangle]\!](\epsilon) = w \cdot o(x) \qquad [\![\langle w, x\rangle]\!](aw_1) = w \cdot ([\![t(x)(a)]\!](w_1))$$

4.1.2.4 Probabilistic Moore automata

Take $T(X) = \mathcal{D}_\omega(X)$, η the Dirac distribution (defined below) and, for $f\colon X \to T(Y)$, $f^\sharp\colon T(X) \to T(Y)$ is given by

$$f^\sharp(c) = \lambda y. \sum_{d \in \mathcal{D}_\omega(Y)} \left(\sum_{x \in f^{-1}(d)} c(x) \right) \times d(y) \qquad \eta(x) = \lambda x'. \begin{cases} 1 & x = x' \\ 0 & \text{otherwise} \end{cases}$$

Take an FT-coalgebra $\langle o, t\rangle \colon X \to \mathcal{D}_\omega(B) \times \mathcal{D}_\omega(X)^A$. For $B = 2$ (note that $\mathcal{D}_\omega(2) \cong [0,1]$) this gives rise to a *(Rabin) probabilistic automaton* [21]: each state x has an output value in $o(x) \in [0,1]$ and, for each input a, $t(x)(a)$ is a probability distribution of next states. The behaviour of a state x is given by $[\![\eta(x)]\!] \colon A^* \to [0,1]$, defined below. Intuitively, one can think of $[\![\eta(x)]\!]$ as a probabilistic language: each word is associated with a value $p \in [0,1]$.

$$\begin{aligned} [\![d]\!](\epsilon) &= \sum_{b\in[0,1]} \Big(\sum_{o(x)=b} d(x)\Big) \times b \\ [\![d]\!](aw) &= [\![\lambda x'. \sum_{c\in\mathcal{D}_\omega(X)} (\textstyle\sum_{b=t(x)(a)} d(x)) \times c(x')]\!](w) \end{aligned}$$

It is worth to note that this exactly captures the semantics of [21], while the ordinary $\sim_{FT}$ coincides with *probabilistic bisimilarity* of [14].

5 Coalgebras and T-Algebras

In the previous section we presented a framework, parameterized by a functor F and a monad $\mathbf{T}$, in which systems of type FT (that is, FT-coalgebras) can be studied using a novel equivalence $\approx_F^T$ instead of the classical $\sim_{FT}$. The only requirement we imposed was that $FT(X)$ has to be a $\mathbf{T}$-algebra.

In this section, we will present functors F for which the requirement of $FT(X)$ being a $\mathbf{T}$-algebra is guaranteed because they can be *lifted* to a functor F^* on $\mathbf{T}$-algebra. For these functors, the equivalence $\approx_F^T$ coincides with $\sim_{F^*}$. In other words, working on FT-coalgebras in $\mathbf{Set}$ under the novel $\approx_F^T$ equivalence is the same as working on F^*-coalgebras on $\mathbf{T}$-algebras under the ordinary $\sim_{F^*}$ equivalence. Next, we will prove that for this class of functors and an arbitrary monad $\mathbf{T}$ the equivalence $\sim_{FT}$ is contained in $\approx_F^T$. Instantiating this result for our first motivating example of non-deterministic automata will yield the well known fact that bisimilarity implies trace equivalence.

Let $\mathbf{T}$ be a monad. An endofunctor $F^* \colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}^{\mathbf{T}}$ is said to be the $\mathbf{T}$-*algebra lifting* of a functor $F \colon \mathbf{Set} \to \mathbf{Set}$ if the following square commutes[1]:

$$\begin{array}{ccc} \mathbf{Set}^{\mathbf{T}} & \xrightarrow{F^*} & \mathbf{Set}^{\mathbf{T}} \\ {\scriptstyle U^{\mathbf{T}}}\downarrow & & \downarrow{\scriptstyle U^{\mathbf{T}}} \\ \mathbf{Set} & \xrightarrow{F} & \mathbf{Set} \end{array}$$

If the functor F has a $\mathbf{T}$-algebra lifting F^* then $FT(X)$ is the carrier of the algebra $F^*(T(X), \mu)$. Functors that have a $\mathbf{T}$-algebra lifting are given, for example, by those endofunctors on $\mathbf{Set}$ constructed inductively by the following grammar

$$F ::= Id \mid B \mid F \times F \mid F^A \mid TG$$

where A is an arbitrary set, B is the constant functor mapping every set X to the carrier of a $\mathbf{T}$-algebra (B, h), and G is an arbitrary functor. Since the forgetful functor $U^{\mathbf{T}} \colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ creates and preserves limits, both $F_1 \times F_2$ and F^A have a $\mathbf{T}$-algebra lifting if F, F_1, and F_2 have. Finally, TG has a $\mathbf{T}$-algebra lifting for every endofunctor G given by the assignment $(X, h) \mapsto (TGX, \mu_{GX})$. Note that we do not allow taking coproducts in the above grammar, because coproducts of $\mathbf{T}$-algebras are not preserved in general by the forgetful functor $U^{\mathbf{T}}$. Instead, one could resort to extending the grammar with the carrier of the coproduct taken

[1] This is equivalent to the existence of a distributive law $\lambda \colon TF \Rightarrow FT$ [12].

directly in $\mathbf{Set}^{\mathbf{T}}$. For instance, if $\mathbf{T}$ is the (finite) powerset monad, then we could extend the above grammar with the functor $F_1 \oplus F_2 = F_1 + F_2 + \{\top, \bot\}$.

Now, let F be a functor with a $\mathbf{T}$-algebra lifting and for which a final coalgebra Ω exists. If Ω can be constructed as the limit of the final sequence (for example assuming the functor accessible [1]), then, because the forgetful functor $U^{\mathbf{T}} \colon \mathbf{Set}^{\mathbf{T}} \to \mathbf{Set}$ preserves and creates limits, Ω is the carrier of a $\mathbf{T}$-algebra, and it is the final coalgebra of the lifted functor F^*. Further, for any FT-coalgebra $f \colon X \to FT(X)$, the unique F-coalgebra homomorphism $[\![-]\!]$ as in diagram (1) is a T-algebra homomorphism between $T(X)$ and Ω. Conversely, the carrier of the final F^*-coalgebra (in $\mathbf{Set}^{\mathbf{T}}$) is the final F-coalgebra (in $\mathbf{Set}$).

Intuitively, the above means that for an accessible functor F with a $\mathbf{T}$-algebra lifting F^*, F^*-equivalence in $\mathbf{Set}^{\mathbf{T}}$ coincides with F-equivalence with respect to $\mathbf{T}$ in $\mathbf{Set}$. The latter equivalence is coarser than the FT-equivalence in $\mathbf{Set}$, as stated in the following theorem.

▶ **Theorem 1.** *Let $\mathbf{T}$ be a monad. If F is an endofunctor on $\mathbf{Set}$ with a $\mathbf{T}$-algebra lifting, then $\sim_{FT}$ implies $\approx_F^T$.*

The proof of this theorem (presented in [27]) relies on the fact that for every monad $\mathbf{T}$ and functor F with a $\mathbf{T}$-algebra lifting, if $h \colon (X, f) \to (Y, g)$ is an FT-coalgebra homomorphism, then $(\eta_Y \circ h)^\sharp \colon (T(X), f^\sharp) \to (T(Y), g^\sharp)$ is an F-coalgebra homomorphism.

The above theorem instantiates to the well-known facts: for NDA, where $F(X) = 2 \times X^A$ and $T = \mathcal{P}_\omega$, that bisimulation implies language equivalence; for partial automata, where $F(X) = 2 \times X^A$ and $T = 1 + -$, that equivalence of pairs of languages, consisting of defined paths and accepted words, implies equivalence of accepted words; for probabilistic automata, where $F(X) = [0, 1] \times X^A$ and $T = \mathcal{D}_\omega$, that probabilistic bisimilarity implies probabilistic/weighted language equivalence. Note that, in general, the above inclusion is strict.

6 Discussion

In this paper, we lifted the powerset construction on automata to the more general framework of FT-coalgebras. Our results lead to a uniform treatment of several kinds of existing and new variations of automata (that is, FT-coalgebras) by an algebraic structuring of their state space through a monad T. We showed as examples partial Mealy machines, structured Moore automata, nondeterministic, partial and probabilistic automata. The technical report [27] shows (as further examples) several behavioural equivalences that are extremely interesting for the theory of concurrency. It is worth mentioning that the framework instantiates to many other examples, among which *weighted automata* [26]. These are simply structured Moore automata for $B = 1$ and $\mathbf{T} = \mathbb{S}_\omega^-$ (for a semiring $\mathbb{S}$) [7]. It is easy to see that $\sim_{FT}$ coincides with weighted bisimilarity [5], while $\approx_F^T$ coincides with weighted language equivalence [26].

Some of the aforementioned examples can also be coalgebraically characterized in the framework of [9]. There, instead of considering FT-coalgebras on $\mathbf{Set}$ and F^*-coalgebras on $\mathbf{Set}^{\mathbf{T}}$ (the Eilenberg-Moore category), TG-coalgebras on $\mathbf{Set}$ and $\overline{G}$-coalgebras on $\mathbf{Set}_{\mathbf{T}}$ (the *Kleisli* category) are studied. The main theorem of [9] states that under certain assumptions, the initial G-algebra is the final $\overline{G}$-coalgebra that characterizes (generalized) trace equivalence. In [27], we present a first step in exploring the connection between both frameworks. However, the exact relationship is not clear yet and further research is needed in order to make it precise. It is worth to remark that many of our examples will not fit the framework in [9]: for instance, the exception, the side effect, the full-probability and the interactive output monads do not fulfill their requirements (the first three do not have a bottom element and

the latter is not commutative). Moreover, we also note that the example of partial Mealy machines is not purely trace-like, as all the examples in [9].

There are two other future research directions. On the one hand, we will try to exploit *F-bisimulations up to T* [15, 16] as a sound and complete proof technique for $\approx_F^T$. On the other hand, we would like to lift many of those coalgebraic tools that have been developed for "branching equivalences" (such as coalgebraic modal logic [6, 25] and (axiomatization for) regular expressions [3]) to work with the "linear equivalences" induced by $\approx_F^T$.

References

1 J. Adámek. Free algebras and automata realization in the language of categories. *Comment. Math. Univ. Carolinae*, 15:589–602, 1974.

2 Falk Bartels. *On generalized coinduction and probabilistic specification formats*. PhD thesis, Vrije Universiteit Amsterdam, 2004. PhD thesis.

3 M. M. Bonsangue, J. J. M. M. Rutten, and A. Silva. An algebra for Kripke polynomial coalgebras. In *LICS*, pages 49–58. IEEE Computer Society, 2009.

4 S. D. Brookes, C. A. R. Hoare and A. W. Roscoe. A Theory of Communicating Sequential Processes. *J. ACM.*, 31(3):560–599, 1984.

5 P. Buchholz. Bisimulation relations for weighted automata. *TCS*, 393(1-3):109–123, 2008.

6 C. Cîrstea, A. Kurz, D. Pattinson, L. Schröder, and Y. Venema. Modal logics are coalgebraic. In Erol Gelenbe, Samson Abramsky, and Vladimiro Sassone, editors, *BCS Int. Acad. Conf.*, pages 128–140. British Computer Society, 2008.

7 H. P. Gumm and T. Schröder. Monoid-labeled transition systems. *ENTCS*, 44(1), 2001.

8 H.H. Hansen. Coalgebraising subsequential transducers. *ENTCS*, 203(5):109–129, 2008.

9 I. Hasuo, B. Jacobs, and A. Sokolova. Generic trace semantics via coinduction. *Logical Methods in Computer Science*, 3(4), 2007.

10 C. A. R. Hoare. Communicating Sequential Processes. *Commun. ACM.*, 21(8):666–677, 1978.

11 Bart Jacobs. Distributive laws for the coinductive solution of recursive equations. Inf. Comput. 204(4): 561-587 (2006)

12 P.T. Johnstone. Adjoint lifting theorems for categories of algebras. *Bulletin London Mathematical Society*, 7:294–297, 1975.

13 C. Jou and S. A. Smolka. Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes. In Proc. of CONCUR, *LNCS*, 458:367–383. Springer, 1990.

14 K. G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Inf. Comput.*, 94(1):1–28, 1991.

15 M. Lenisa. From Set-theoretic Coinduction to Coalgebraic Coinduction: some results, some problems. *ENTCS*, 19, 1999.

16 M. Lenisa, J. Power and H. Watanabe. Distributivity for endofunctors, pointed and copointed endofunctors, monads and comonads. *ENTCS*, 33, 2000.

17 E.Manes. Algebraic theories. *Graduate Texts in Mathematics*, 26, 1976.

18 E. Moggi. Notions of computation and monads. *Inf. Comput.*, 93(1):55–92, 1991.

19 L. Monteiro. A Coalgebraic Characterization of Behaviours in the Linear Time - Branching Time Spectrum. In Proc. of WADT, *LNCS*, 5486:128–140. Springer, 2009.

20 E.-R. Olderog and C. A. R. Hoare. Specification-Oriented Semantics for Communicating Processes. *Acta Inf.*, 21(1):9–66, 1986.

21 M. O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.

22 J. J. M. M. Rutten. Universal coalgebra: a theory of systems. *TCS*, 249(1):3–80, 2000.

23 J. J. M. M. Rutten. Algebraic specification and coalgebraic synthesis of mealy automata. *ENTCS*, 160:305–319, 2006.

24 J.J.M.M. Rutten. Coalgebra, concurrency, and control. In R. Boel and G. Stremersch, editors, *Proceedings of WODES 2000*, pages 31–38, 2000.

25 L. Schröder. Expressivity of coalgebraic modal logic: The limits and beyond. *TCS*, 390(2-3):230–247, 2008.

26 M. P. Schützenberger. On the definition of a family of automata. *Information and Control*, 4(2-3):245–270, 1961.

27 A. Silva, F. Bonchi, M. Bonsangue and J. Rutten. Generalizing the powerset construction, coalgebraically. Technical Report. Nr. SEN-1008, September 2010, CWI.

Uniqueness of Normal Forms is Decidable for Shallow Term Rewrite Systems*

Nicholas Radcliffe[1] and Rakesh M. Verma[2]

1 **Computer Science Department**
Virginia Tech
114 McBryde Hall, Blacksburg, VA 24061, USA
nradclif@vt.edu

2 **Computer Science Department**
University of Houston
501 Philip G. Hoffman Hall, Houston, TX 77204, USA
rmverma@cs.uh.edu

Abstract

Uniqueness of normal forms ($UN^{=}$) is an important property of term rewrite systems. $UN^{=}$ is decidable for ground (i.e., variable-free) systems and undecidable in general. Recently it was shown to be decidable for linear, shallow systems. We generalize this previous result and show that this property is decidable for shallow rewrite systems, in contrast to confluence, reachability and other properties, which are all undecidable for flat systems. Our result is also optimal in some sense, since we prove that the $UN^{=}$ property is undecidable for two superclasses of flat systems: left-flat, left-linear systems in which right-hand sides are of depth at most two and right-flat, right-linear systems in which left-hand sides are of depth at most two.

Keywords and phrases term rewrite systems, uniqueness of normal forms, decidability, shallow rewrite systems, flat rewrite systems

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.284

1 Introduction

Term rewrite systems (TRSs), finite sets of rules, are useful in many computer science fields including theorem proving, rule-based programming, and symbolic computation. An important property of TRSs is confluence (also known as the Church-Rosser property), which implies uniqueness of normal forms ($UN^{=}$). Normal forms are expressions to which no rule is applicable. A TRS has the $UN^{=}$ property if there are *not* distinct normal forms n, m such that $n \overset{*}{\leftrightarrow}_R m$, where $\overset{*}{\leftrightarrow}_R$ is the symmetric closure of the rewrite relation induced by the TRS R.

Uniqueness of normal forms is an interesting property in itself and well-studied [9]. Confluence can be a requirement too strong for some applications such as lazy programming. Additionally, in the proof-by-consistency approach for inductive theorem proving, consistency is often ensured by requiring the $UN^{=}$ property.

We study the decidability of uniqueness of normal forms. Uniqueness of normal forms is decidable for ground systems [12], but is undecidable in general [12]. Since the property is undecidable in general, we would like to know for which classes of rewrite systems, beyond ground systems, we can decide $UN^{=}$. In [13, 14] a polynomial time algorithm for this property was given for linear, shallow rewrite systems. A rewrite system is *linear* if variables occur at

* Research supported in part by NSF grants CCF-0306475, DUE-0755500, and DUE-0737404.

most once in each side of any rule. It is *shallow* if variables occur only at depth zero or depth one in each side of any rule. It is *flat* if both the left- and right-hand sides of all the rules have height zero or one. An example of a linear flat (in fact, ground) system that has $\text{UN}^=$ but not confluence is $\{f(c) \to 1,\ c \to g(c)\}$. More sophisticated examples can be constructed using a sequential 'or' function in which the second argument gives rise to a nonterminating computation.

In this paper, we consider the class of shallow systems, i.e., we drop the linearity restriction of [13], and a subset of this class, the flat systems. For flat systems many properties are known to be undecidable including confluence, reachability, joinability, and existence of normal forms [7, 11, 3]. On the other hand, the word problem is known to be decidable for shallow systems [1]. This paper shows that the uniqueness of normal forms problem is decidable for the class of shallow term rewrite systems, which is a significant generalization of [13] and also somewhat surprising since so many properties are undecidable for this class of systems. We also prove the undecidability of $\text{UN}^=$ for two subclasses of linear systems: left-hand sides are linear, flat and right-hand sides are of depth at most two and conversely right-flat, right-linear, and depth two left-hand sides, which shows that our result is optimal as far as depth restrictions are involved and close to optimal as far as linearity and depth restrictions are concerned (the problem is undecidable for the linear, depth-two class [11]).

The structure of our decidability proof is as follows: in [13, 14] it was shown that $\text{UN}^=$ for shallow systems can be reduced to $\text{UN}^=$ for flat systems, (ii) checking $\text{UN}^=$ for flat systems can be reduced to searching for equational proofs between terms drawn from a finite set of terms, and (iii) existence of equational proofs between terms in part (ii) is done thanks to the decidability of the word problem by Comon et al. [1].

Our strategy for part (ii) above, assuming a flat TRS, R, is to show that a sufficiently small witness to non-$UN^=$ for R exists if, and only if, any witness at all exists. To see this, say $\langle M, N\rangle$ is a minimal witness to non-$UN^=$ (in that the sum of the sizes of M and N is minimal). We show that we can replace certain subterms of M and N that are not equivalent to constants with variables, obtaining a witness $\langle M', N'\rangle$. If the heights of M' and N' are both strictly less than the maximum of $\{1, C\}$, where C is the number of constants in our rewrite system, then $\langle M', N'\rangle$ is sufficiently small. Otherwise, M' or N' must have a big subterm (i.e. a subterm whose height is greater than, or equal to, the number of constants), and this subterm is equivalent to a constant. However, in this case (when there is a constant that is equivalent to a big subterm of a component of a minimal witness), we can show that there is a small witness to non-$UN^=$. So, in all cases, we end up with a small witness.

Comparison with related work. Viewed at a very high level, the proof of decidability shows some similarity with other decidability proofs of properties of rewrite systems such as [2]. The basic insight seems to be that, just as in algebra the terms that equal 0 are crucial in a sense, so in rewriting are the terms that reduce to (or are equivalent to) constants. Of course, this observation is about as helpful in proofs of decidability as a compass is to someone lost in a maze. The details in both scenarios are vital and there are many twists and turns. The proof of undecidability shows some similarity with proofs in [12, 4].

1.1 Definitions

Terms. A *signature* is a set $\mathcal{F}$ along with a function *arity*: $\mathcal{F} \to \mathbb{N}$. Members of $\mathcal{F}$ are called *function symbols*, and $arity(f)$ is called the *arity* of the function symbol f. Function symbols of arity zero are called *constants*. Let X be a countable set disjoint from $\mathcal{F}$ that we shall call the set of *variables*. The set $\mathcal{T}(\mathcal{F}, X)$ of *$\mathcal{F}$-terms over X* is defined to be the smallest set that contains X and has the property that $f(t_1, \ldots, t_n) \in \mathcal{T}(\mathcal{F}, X)$ whenever

$f \in \mathcal{F}$, $n = arity(f)$, and $t_1, \ldots, t_n \in \mathcal{T}(\mathcal{F}, X)$. The set of function symbols with arity n is denoted by $\mathcal{F}_n$; in particular, the set of constants is denoted by $\mathcal{F}_0$. We use $root(t)$ to refer to the outermost function symbol of t.

The *size*, $|t|$, of a term t is the number of occurrences of constants, variables and function symbols in t. So, $|t| = 1$ if t is a constant or a variable, and $|t| = 1 + \Sigma_{i=1}^{n}|t_i|$ if $t = f(t_1, \ldots, t_n)$ for $n > 0$. The *height* of a term t is 0 if t is a constant or a variable, and $1 + max\{height(t_1), \ldots, height(t_n)\}$ if $t = f(t_1, \ldots, t_n)$. If a term t has height zero or one, then it is called $flat$. A *position* of a term t is a sequence of natural numbers that is used to identify the locations of subterms of t. The subterm of $t = f(t_0, \ldots, t_{n-1})$ at position p, denoted $t|_p$, is defined recursively: $t|_\lambda = t$, $t|_k = t_k$, for $0 \leq k \leq n-1$, and $t|_{k.p} = (t|_k)|_p$. If $t = f(t_0, \ldots, t_{n-1})$, then we call $t_0, \ldots, t_{n-1}$ the *depth*-1 subterms of t. If all variables appearing in t are either t itself or depth-1 subterms of t, then we say that t is *shallow*. The notation $g[a]$ focuses on (any) one occurrence of subterm a of term g, and $s\{u \mapsto v\}$ denotes the term obtained from term s by replacing all occurrences of the subterm u in s by term v.

A *substitution* is a mapping $\sigma : X \to \mathcal{T}(\mathcal{F}, X)$ that is the identity on all but finitely many elements of X. Substitutions are generally extended to a homomorphism on $\mathcal{T}(\mathcal{F}, X)$ in the following way: if $t = f(t_1, \ldots, t_k)$, then (abusing notation) $\sigma(t) = f(\sigma(t_1), \ldots, \sigma(t_k))$. Oftentimes, the application of a substitution to a term is written in postfix notation. A $unifier$ of two terms s and t is a substitution σ (if it exists) such that $s\sigma = t\sigma$. We assume familiarity with the concept of *most general unifier* [9], which is unique up to variable renaming and denoted by mgu.

Term Rewrite Systems. A *rewrite rule* is a pair of terms, (l, r), usually written $l \to r$. For the rule $l \to r$, the *left-hand side* is $l \notin X$, and the *right-hand side* is r. Notice that l cannot be a variable. A rule, $l \to r$, can be applied to a term, t, if there exists a substitution, σ, such that $l\sigma = t'$, where t' is a subterm of t; in this case, t is rewritten by replacing the subterm $t' = l\sigma$ with $r\sigma$. The process of replacing the subterm $l\sigma$ with $r\sigma$ is called a *rewrite*. A *root rewrite* is a rewrite where $t' = t$. A rule $l \to r$ is *flat* (resp. shallow) if both l and r are flat (resp. shallow). The rule $l \to r$ is *collapsing* if r is a variable. A *term rewrite system* (or *TRS*) is a pair, $(\mathcal{T}, R)$, where R is a finite set of rules and $\mathcal{T}$ is the set of terms over some signature. A TRS, R, is *flat* (resp. shallow) if all of the rules in R are flat (resp. shallow). If we think of $\to$ as a relation, then $\overset{+}{\to}$ and $\overset{*}{\to}$ denote its transitive closure, and reflexive and transitive closure, respectively. Also, $\leftrightarrow$, $\overset{+}{\leftrightarrow}$, and $\overset{*}{\leftrightarrow}$ denote the symmetric closure, symmetric and transitive closure, and symmetric, transitive, and reflexive closure, respectively. We put an 'r' over arrows to denote a root rewrite, i.e., $\overset{r}{\leftrightarrow}$.

A *derivation* is a sequence of terms, $t_1, \ldots, t_n$, such that $t_i \to t_{i+1}$ for $i = 1, \ldots, n-1$; this sequence is often denoted by $t_1 \to t_2 \to \ldots \to t_n$. A *proof* is a sequence, $t_1, \ldots, t_n$, such that $t_i \leftrightarrow t_{i+1}$ for $i = 1, \ldots, n-1$; this sequence is generally denoted by $t_1 \leftrightarrow t_2 \leftrightarrow \ldots \leftrightarrow t_n$. If R is a rewrite system, then a proof is *over* R if it can be constructed using rules in R. If π is a proof, we say that $\pi \in s\overset{*}{\leftrightarrow}t$ if π is of the form $s \leftrightarrow \ldots \leftrightarrow t$ (it is possible for the proof sequence to consist of a single term, in which case $s = t$ and the proof is simply a sequence with a single element, s). We say that $\pi \in s\overset{+}{\leftrightarrow}t$ if $\pi \in s\overset{*}{\leftrightarrow}t$ and the proof sequence contains at least two terms. We write $s\overset{*}{\leftrightarrow}t$ (resp. $s\overset{+}{\leftrightarrow}t$) to denote that there is a proof, π, with $\pi \in s\overset{*}{\leftrightarrow}t$ (resp. $\pi \in s\overset{+}{\leftrightarrow}t$).

A *normal form* is a term, $t \in \mathcal{T}(\mathcal{F}, X)$, such that no subterm of t can be rewritten. A term that is not a normal form, i.e., one with a subterm that *can* be rewritten, is called *reducible*. We denote the set of all normal forms for R by NF_R, or simply NF. A rewrite system R is $UN^{=}$ if it is *not* the case that R has two distinct normal forms, M and N, such that $M\overset{*}{\leftrightarrow}N$. If such a pair exists, then we say that the pair, $\langle M, N\rangle$, is a *witness* to

non-$UN^{=}$. The *size of a witness*, denoted $|\langle M, N\rangle|$, is $|M| + |N|$. A *minimal witness* is a witness with minimal size. Finally, we define $SubMinWit_R$ to be set of all terms M' such that $\langle M, N\rangle$ is a minimal witness, and M' is a subterm of M.

2 Preliminary Results

We begin with a few simple results, whose proofs are omitted to save space, on when rules apply. They are used throughout the paper to show that normal forms are preserved under certain transformations. Before we begin, notice that it is relatively simpler to preserve normal forms when the relevant TRS is linear. For instance, imagine any *flat and linear* TRS such that $f(g(a), h(b))$ is a normal form. Since $g(a)$ is evidently a normal form, $f(g(a), g(a))$ would also be a normal form, when the TRS is linear. If the TRS is not linear, then there could be a rule of the form $f(x, x) \to t$, making $f(g(a), g(a))$ reducible. The results below handle such complications presented by non-linear rules.

▶ **Definition 1.** Let R be a rewrite system, and let $l \to r = \rho \in R$ be a rule. The *pattern of* ρ, denoted $Patt(\rho)$, is a set of equations $\{i = j \mid \ l|_i = l|_j, l|_i, l|_j \in X\}$.

▶ **Definition 2.** Let $t \in \mathcal{T}(\mathcal{F}, X)$ be a term with $root(l) = root(t)$. If $A = \{i_1, i_2, \ldots, i_k\}$ is the set of positions that appear in equations in $Patt(\rho)$, then the *pattern of* t *with respect to* ρ, denoted $Patt_\rho(t)$, is the set $\{i_a = i_b \mid t|_{i_a} = t|_{i_b}, i_a, i_b \in A\}$.

Note that $Patt_\rho(t)$ is undefined if $root(l) \neq root(t)$.

▶ **Lemma 3.** *Let R be a flat TRS. Let $t \in \mathcal{T}(\mathcal{F}, X)$ be a term, and let $l \to r = \rho \in R$ be a rule. Then ρ can be applied to t at λ if, and only if, (i) $l|_i = t|_i$ whenever $l|_i$ is a constant, and (ii) $Patt_\rho(t)$ is defined and $Patt(\rho) \subseteq Patt_\rho(t)$.*

Consider the term $f(a, x, x, g(b))$. Let's assume that it is a normal form. We want to know if altering depth-1 subterms can make the term reducible. Clearly, replacing x with a constant could *potentially* make the term reducible, depending on the rules in the rule set. But what about replacing any of the depth-1 subterms with a normal form containing a fresh variable? Notice that such a replacement could not make condition (i) of the above lemma true if it had been false. But what if condition (i) is true and condition (ii) is false? Could replacing a depth-1 subterm, or even several depth-1 subterms, with terms containing fresh variables make condition (ii) true? This question is answered by the following proposition.

▶ **Proposition 4.** *Let R be a flat TRS, and let $M = f(s_1, \ldots, s_m)$ be a normal form for R. Let $S = \{t_{i_1}, \ldots, t_{i_n}\}$ be a set of normal forms, where $n \leq m$ and each term contains at least one fresh variable (relative to M). Further, say that $t_{i_j} \neq t_{i_k}$ whenever $s_{i_j} \neq s_{i_k}$ for all $i_j, i_k \in \{i_1, \ldots, i_n\}$. If M' is what one obtains from M by replacing each s_{i_j} with t_{i_j}, then $M' \in NF_R$.*

▶ **Lemma 5.** *If R is any TRS such that $f(t_1, \ldots, t_m) \in SubMinWit_R$, then $t_i \overset{*}{\leftrightarrow}_R t_j$ is impossible for $t_i \neq t_j$. This is equivalent to saying that there is no term s that is equivalent to both t_i and t_j via R.*

2.1 Normal Forms Equivalent to Constants

Let E be a finite set of equations. Following the authors of [1], we extend E to $\widehat{E}$ by closing under the following inference rules:

1. $\dfrac{g = d,\ l = r}{d\sigma = r\sigma}$ if $l, g \notin X$ and $\sigma = mgu(l, g)$

2. $\dfrac{x = d,\ y = r}{d = r\{y \mapsto x\}}$ if $y \in X$ and $x \in \mathcal{F}_0 \cup X$
3. $\dfrac{g[a] = d,\ a = b}{g[b] = d}$ if $a, b \in \mathcal{F}_0$

Notice that if E is flat, then $\widehat{E}$ is flat, as well.

We can think of a rewrite system as a set of equations: if $s \to t$ is a rule in R, then $s \leftrightarrow t$ is its corresponding equation. We write E_R for the set of equations obtained in this way from a rewrite system R. Clearly, if s and t are terms in $\mathcal{T}(\mathcal{F}, X)$, then they are R-equivalent if and only if they are E_R equivalent. Also, from [1] we know that terms are E_R equivalent if, and only if, they are $\widehat{E_R}$-equivalent. In [1], the authors show that, if R is a shallow TRS and $s, t \in \mathcal{T}(\mathcal{F}, X)$, then there is a procedure that produces, for any proof, $\pi \in s \overset{*}{\leftrightarrow}_R t$, over R, a new proof, which is denoted by $\pi_{1rr} \in s \overset{*}{\leftrightarrow}_{\widehat{E_R}} t$, over $\widehat{E_R}$, such that there is at most one root rewrite step in π_{1rr}.

Consider the following example: $R = \{f(x,x) \to c, f(x,x) \to g(a,x), g(a,x) \to g(a,x), a \to h(b), b \to h(c)\}$. It is easy to check that $\widehat{E_R} = E_R \cup \{c \leftrightarrow g(a,x)\}$. We use $\widehat{E_R}$ to search for a minimal witness to non-$UN^{=}$ for R; in particular, we will use the fact that for every proof $s \overset{*}{\leftrightarrow}_R t$, there is a proof $s \overset{*}{\leftrightarrow}_{\widehat{E_R}} t$ with at most one root rewrite.

Clearly, c is an R-normal form, so if we are looking for a minimal witness to non-$UN^{=}$ for R, $\langle c, ? \rangle$ might be a good first guess. We know that $c \leftrightarrow_{\widehat{E_R}} f(x,x)$, so maybe $\langle c, f(u,v) \rangle$ is a minimal witness, for some normal forms u and v. This is not possible. First, notice that $f(x,x)$ appears on the LHS of a rule, so $f(t,t)$ cannot be a normal form, for arbitrary term t. Second, notice that if $f(t,t)$ is equivalent to another normal form, then we can assume it is of the form $f(u,v)$, because we have already "used up" our only root rewrite by using $c \leftrightarrow_{\widehat{E_R}} f(x,x)$. So, maybe we can plug some term, t, into x, and then rewrite one instance of it to a normal form u, and another instance of it to a normal form v, obtaining a minimal witness of the form $\langle c, f(u,v) \rangle$? This cannot be the case, because if $\langle c, f(u,v) \rangle$ is a minimal witness, then (by Lemma 5 and the fact that $u \overset{*}{\leftrightarrow} v$) $\langle u, v \rangle$ would violate the minimality of $\langle c, f(u,v) \rangle$. So, we should consider $c \leftrightarrow_{\widehat{E_R}} g(a,x)$ as *the* (one and only) rewrite step in our proof. We know that a is not a normal form, and must, therefore, be rewritten to one - $h(h(c))$. But what about x? Should we plug anything into it? Say we were to plug t into x, and then rewrite t to some normal form, u. This would be unnecessary, because non-linearity is not an issue here, and so we can leave x as it is. So, $\langle c, g(h(h(c)), x) \rangle$ is a minimal witness, and the relevant proof looks like: $c \leftrightarrow_{\widehat{E_R}} g(a,x) \leftrightarrow_{\widehat{E_R}} g(h(b),x) \leftrightarrow_{\widehat{E_R}} g(h(h(c)),x)$.

Now, here is the interesting part. Notice that we have *four* R-normal forms equivalent to constants, but only *three* constants in R, i.e, $c \overset{*}{\leftrightarrow}_{\widehat{E_R}} c$, $h(c) \overset{*}{\leftrightarrow}_{\widehat{E_R}} b$, $h(h(c)) \overset{*}{\leftrightarrow}_{\widehat{E_R}} a$, and $g(h(h(c)),x) \overset{*}{\leftrightarrow}_{\widehat{E_R}} c$. From the Pigeonhole Principle, we can conclude that there must be some constant in R that is equivalent to two distinct normal forms (of course, we already knew this, but in general this technique will be useful). We generalize the lessons learned from this example in the following results.

▶ **Lemma 6.** *Let R be a flat TRS. Let $\langle M_0, M_1 \rangle$ be a minimal witness to non-$UN^{=}$ for R, and say $M = f(t_1, \ldots, t_m)$ is a subterm of M_0. Let c be a constant, and let $c \overset{r}{\leftrightarrow}_{\widehat{E_R}} f(s_1, \ldots, s_m) \overset{*}{\leftrightarrow}_{\widehat{E_R}} f(t_1, \ldots, t_m) = M$ be a proof with a single root rewrite. If s_i is not a constant, then $height(t_i) = 0$.*

Proof. Let S_{const} be the set of positive integers, i, such that $s_i \in \mathcal{F}_0$. If none of the s_i's is a variable, then there is nothing to show; so, assume at least one of the s_i's is a variable. Now,

let

$$s'_j = \begin{cases} s_j & \text{if } j \in S_{const} \\ x_{s_j} & \text{otherwise} \end{cases} \quad \text{and} \quad t'_j = \begin{cases} t_j & \text{if } j \in S_{const} \\ x_{s_j} & \text{otherwise} \end{cases}$$

where x_{s_j} is a fresh variable not appearing in M_0 or M_1, and $x_{s_i} = x_{s_j}$ if and only if $s_i = s_j$. We show that (i) $f(s'_1, \ldots, s'_m) \overset{*}{\leftrightarrow}_{\widehat{E_R}} f(t'_1, \ldots, t'_m)$, (ii) $f(t'_1, \ldots, t'_m) \in NF_R$, and (iii) for $i \notin S_{const}$, $height(t_i) = 0$.

Part (i). If $j \notin S_{const}$, then $s'_j = t'_j = x_{s_j}$. So, say $j \in S_{const}$. In this case, $s'_j = s_j \overset{*}{\leftrightarrow}_{\widehat{E_R}} t_j = t'_j$. So, $f(s'_1, \ldots, s'_m) \overset{*}{\leftrightarrow}_{\widehat{E_R}} f(t'_1, \ldots, t'_m)$. *Part (ii).* Let $j, j' \notin S_{const}$, and say $t_j \neq t_{j'}$. In order to apply Proposition 4, we need to show that $t'_j \neq t'_{j'}$. From Lemma 5, we know that $s_j \neq s_{j'}$, and hence $t'_j = x_{s_j} \neq x_{s_{j'}} = t'_{j'}$. Therefore, we can apply Proposition 4 to obtain that $f(t'_1, \ldots, t'_m) \in NF_R$. *Part (iii).* Notice that, by (i) and $f(s'_1, \ldots, s'_m) \overset{*}{\leftrightarrow}_{\widehat{E_R}} c$, we have $f(t_1, \ldots, t_m) \overset{*}{\leftrightarrow}_{\widehat{E_R}} c \overset{*}{\leftrightarrow}_{\widehat{E_R}} f(t'_1, \ldots, t'_m) = N$. Also, since N contains at least one fresh variable not appearing in M_0 or M_1, we know that $M \neq N$ and $C[N] \neq M_0$ or M_1, where $C[]$ is a context and $M_0 = C[M]$. Hence $\langle C[N], M_1 \rangle$ is a witness to non-$UN^=$, with $|C[N]| \leq |M_0|$. But $\langle M_0, M_1 \rangle$ is a minimal witness, so $|C[N]| = |C[M]|$ and $|N| = |M|$. Since $|t'_i| = 1$ for all $i \notin S_{const}$, it must be the case that $|t_i| = 1$. Thus, we have that $height(t_i) = height(t'_i) = 0$ for all $i \notin S_{const}$. ■

▶ **Corollary 7.** *Under the same assumptions as Lemma 6 plus the assumption that at least one of the s_i's is a constant, there is a j such that $s_j \in \mathcal{F}_0$ and $height(t_j) = height(f(t_1, \ldots, t_m)) - 1$ with $1 \leq j \leq m$.*

Proof. Since $height(t_i) = 0$ whenever $s_i \notin \mathcal{F}_0$, we know that $height(t_i) \leq height(t_j)$ whenever $s_i \notin \mathcal{F}_0$ and $s_j \in \mathcal{F}_0$. So, amongst the direct subterms of $f(t_1, \ldots, t_m)$ with maximal height, there must be one, t_j, such that $s_j \in \mathcal{F}_0$. ■

▶ **Proposition 8.** *Let R be a flat TRS, and let $c \in \mathcal{F}_0$. Let $\langle M, N \rangle$ be a minimal witness, and let N' be a subterm of N such that $height(N') = k$. Further, let $\pi \in c \overset{*}{\leftrightarrow} N'$ be a proof over R. Then we can find either (i) $1 + k$ distinct normal forms equivalent to constants, the normal forms having heights $0, 1, \ldots, k$, or (ii) a witness, $\langle N_0, N_1 \rangle$, to non-$UN^=$, such that N_0 and N_1 are flat.*

Proof. We proceed by induction on $height(N')$. For the base case we assume that $height(N') = 0$. If the proof is trivial, i.e., if $c = N'$, then we have $1 = 1 + height(N')$ normal form (with height zero) equivalent to a constant. So, assume that π has at least one step.

We know that there is a proof, $\pi_{1rr} \in c \overset{+}{\leftrightarrow}_{\widehat{E_R}} N'$, such that there is only one root rewrite step in π_{1rr}. Since the first step in π_{1rr} is necessarily a root rewrite, π_{1rr} must have the form $c \overset{r}{\leftrightarrow} w\sigma = N'$, where the rule applied is $c \to w$ or $w \to c$, and $height(w) = 0$ (notice that if $c \overset{r}{\leftrightarrow} u \overset{*}{\leftrightarrow} N'$ for some term u with $height(u) > 0$, then we would need a second root rewrite to get back to N'). If $w \in X$, then $x \leftrightarrow c \leftrightarrow y$, where x, y are distinct variables. Therefore, $\langle x, y \rangle$ is a witness to non-$UN^=$ with x and y flat. If $w \in \mathcal{F}_0$, then we have found $1 = 1 + height(N')$ normal form (with height zero) equivalent to a constant.

For the inductive step, assume that $height(N') > 0$, and that the proposition holds for any height strictly less than $height(N')$. Now, π_{1rr} has the form

$$c \overset{r}{\leftrightarrow}_{\widehat{E_R}} f(t_1, \ldots, t_m) \overset{*}{\leftrightarrow}_{\widehat{E_R}} f(u_1, \ldots, u_m) = N'$$

and $t_i \overset{*}{\leftrightarrow}_{\widehat{E_R}} u_i$ for $1 \leq i \leq m$. We have two cases: (i) there is an i such that $t_i \in \mathcal{F}_0$, and (ii) there is no such i. For (i), by Corollary 7, there exists an i such that t_i is a constant and

$height(u_i) = k - 1$. So, we can apply the inductive hypothesis to conclude that we have either (i) $1 + (1 + (height(N') - 1)) = 1 + height(N')$ distinct normal forms, with heights $0, 1, \ldots, height(N')$, equivalent to constants (the first $height(N') - 1$ normal forms come from the inductive hypothesis, and the final normal form is N' itself, which is equivalent to c), or (ii) a witness, $\langle N_0, N_1 \rangle$, to non-$UN^{=}$, such that N_0 and N_1 are flat.

In case (ii), if $c \leftrightarrow_{\widehat{E_R}} f(s_1, \ldots, s_m)$ is the rule used for $c \leftrightarrow_{\widehat{E_R}} f(t_1, \ldots, t_m)$, then s_i is a variable for $1 \leq i \leq m$. We need to show that $f(s_1, \ldots, s_m) \in NF_R$. From Lemma 5, we know that $t_i \neq t_j$ whenever $u_i \neq u_j$ for $1 \leq i, j \leq m$. Since $t_i \neq t_j$ implies that $s_i \neq s_j$, we see that $s_i \neq s_j$ whenever $u_i \neq u_j$. We can assume that the variables $s_1, \ldots, s_m$ are fresh relative to $f(u_1, \ldots, u_m)$, and so we can replace u_i with s_i in $f(u_1, \ldots, u_m)$, obtaining $f(s_1, \ldots, s_m) \in NF_R$ by Proposition 4. Since $f(s_1, \ldots, s_m)$ is a normal form, we can replace the variables appearing in $f(s_1, \ldots, s_m)$ with fresh variables to produce a new normal form, $f(s'_1, \ldots, s'_m)$, such that $f(s_1, \ldots, s_m) \leftrightarrow_{\widehat{E_R}} c \leftrightarrow_{\widehat{E_R}} f(s'_1, \ldots, s'_m)$. So, $\langle f(s_1, \ldots, s_m), f(s'_1, \ldots, s'_m) \rangle$ is our witness with $f(s_1, \ldots, s_m)$ and $f(s'_1, \ldots, s'_m)$ flat. ■

▶ **Corollary 9.** *Let R be a flat TRS, and let $c \in \mathcal{F}_0$. Let $\langle M, N \rangle$ be a minimal witness, and let N' be a subterm of N, with $height(N') \geq |\mathcal{F}_0|$. Further, let $\pi \in c \overset{*}{\leftrightarrow}_R N'$ be a proof over R. Then we can find either (i) a witness, $\langle M_0, M_1 \rangle$, to non-$UN^{=}$, such that M_0 and M_1 are flat, or (ii) a witness, $\langle N_0, N_1 \rangle$, to non-$UN^{=}$, such that $height(N_0), height(N_1) \leq |\mathcal{F}_0|$.*

Proof. By Proposition 8, we know that we can find either (a) a witness, $\langle M_0, M_1 \rangle$, to non-$UN^{=}$, such that M_0 and M_1 are flat, or (b) $1 + height(N')$ distinct normal forms equivalent to constants. If (a) is the case, then we are done. So assume that (b) is true. Since there are $1 + height(N') > |\mathcal{F}_0|$ normal forms equivalent to, at most, $|\mathcal{F}_0|$ constants, we know, by the Pigeonhole Principle, that a single constant is equivalent to two distinct normal forms. From the above observation, we know that the normal forms have heights 0, 1, 2, …, $height(N')$. The smallest (height-wise) $1 + |\mathcal{F}_0|$ normal forms each have height no more than $|\mathcal{F}_0|$. So, we know that we can find a witness, $\langle N_0, N_1 \rangle$, to non-$UN^{=}$, such that $height(N_0), height(N_1) \leq |\mathcal{F}_0|$. ■

▶ **Proposition 10.** *Let R be a flat TRS. Then, either (i) there does not exist a constant $c \in \mathcal{F}_0$ and normal form $N \in SubMinWit_R$ such that $c \overset{*}{\leftrightarrow}_{\widehat{E_R}} N$ and $height(N) \geq |\mathcal{F}_0|$, or (ii) there exists a witness, $\langle N_0, N_1 \rangle$ to non-$UN^{=}$ for R such that $height(N_0), height(N_1) \leq k = max\{1, |\mathcal{F}_0|\}$. Further, there is an effective procedure to decide whether (i) or (ii) is the case.*

Proof. Consider all ground[1] normal forms over the signature of the rewrite system, i.e., consisting of constants and function symbols appearing in the finitely many rules of R, with height less than, or equal to, k; we use $NF_{\leq k}$ to denote this set. Notice that if there is a constant, $c \in \mathcal{F}_0$, and an element of $SubMinWit_R$, N, with $height(N) \geq |\mathcal{F}_0|$, such that $c \overset{*}{\leftrightarrow} N$, then by Corollary 9 there is a witness, $\langle N_0, N_1 \rangle$, to non-$UN^{=}$ for R with $height(N_0), height(N_1) \leq k$. By a result in [1], the word problem is decidable for flat systems. So, we can construct the set of all pairs, (s, t), such that $s, t \in NF_{\leq k}$ and $s \overset{*}{\leftrightarrow}_R t$. If we do not find a witness to non-$UN^{=}$ in $NF_{\leq k}$, then we know that there is no $c \in \mathcal{F}_0$ and $N \in SubMinWit_R$ such that $height(N) \geq |\mathcal{F}_0|$ and $c \overset{*}{\leftrightarrow}_R N$. Otherwise, we have found the witness $\langle N_0, N_1 \rangle$ with $height(N_0), height(N_1) \leq k$. ■

[1] As in [13, 14], for nonlinear rewrite systems also we can expand the signature of the rewrite system with 3α new constants, where α is the maximum arity of a function symbol in the rules, and focus on ground normal forms.

2.2 Shrinking Witnesses

Say $\langle f(a, g(b, f(c, x))), h(y, y, h(a, b, c))\rangle$ is a witness to non-$UN^{=}$ for some TRS. Can we replace big subterms of a component of the witness, without changing the fact that it is a witness, i.e., if we replace $g(b, f(c, x))$ with a variable, z, will $\langle f(a, z),\ h(y, y, h(a, b, c))\rangle$ still be a witness? We show that we can replace depth-1 subterms that are *not* equivalent to a constant with a variable. This shrinks the size of the witness; in particular, only depth-1 subterms of such a shrunk witness that are equivalent to a constant can have height greater than, or equal to, the number of constants in the TRS. So, a shrunk minimal witness either has small components, or there is a large subterm of a component of a minimal witness that is equivalent to a constant. If the latter is the case, then we know, by Corollary 9, that there is a small witness.

▶ **Definition 11.** Let R be a rewrite system. For each term (up to renaming of variables), t, we can add a new variable $x_{\overline{t}}$ to X without altering the relation $\overset{*}{\leftrightarrow}$, where $x_{\overline{s}} = x_{\overline{t}}$ if, and only if, $s \overset{*}{\leftrightarrow}_R t$. Let $t = f(t_1, \ldots, t_n)$ be a term in $\mathcal{T}(\mathcal{F}, X)$. Then we define

$$\phi(t) = \begin{cases} x_{\overline{t}} & \text{if } t \text{ is not equivalent to a constant} \\ t & \text{otherwise} \end{cases}$$

Let $u = f(u_1, \ldots, u_m)$ for $m > 0$ and $v \in X$. We define the function α that maps terms to terms as follows: $\alpha(u) = f(\phi(u_1), \ldots, \phi(u_m))$ and $\alpha(v) = v$.

Notice that $\alpha(c) = c$ for $c \in \mathcal{F}_0$, since α only affects depth-1 subterms.

▶ **Lemma 12.** *Let R be a flat TRS, and let $u \leftrightarrow_R v$ be a proof over R, where $u \leftrightarrow_R v$ is not a root rewrite. Then, there is a proof $\alpha(u) \overset{*}{\leftrightarrow}_R \alpha(v)$.*

Proof. Say $u = f(u_1, \ldots, u_m)$ and $v = f(v_1, \ldots, v_m)$ (notice that if $u \leftrightarrow_R v$ is not a root rewrite, then neither u nor v can have height zero). Since the rewrite is not a root rewrite, we know that there are u_i and v_i such that $u_i \leftrightarrow_R v_i$, and $u_j = v_j$ for all $j \neq i$. If u_i, v_i are equivalent to a constant, then $\phi(u_i) = u_i$ and $\phi(v_i) = v_i$, and hence $\alpha(u) \leftrightarrow_R \alpha(v)$. If u_i, v_i are not equivalent to a constant, then $\phi(u_i) = x_{\overline{u_i}} = x_{\overline{v_i}} = \phi(v_i)$, and hence $\alpha(u) = \alpha(v)$. ■

▶ **Lemma 13.** *Let R be a flat TRS, and let $u \leftrightarrow_R v$ be a proof over R, where $u \leftrightarrow_R v$ is a root rewrite. If the rewrite has the form $u = w\sigma \to x\sigma = v$ (i.e. it uses a collapsing rule $w \to x$), then $\alpha(u) \leftrightarrow_R \phi(v)$; otherwise $\alpha(u) \leftrightarrow_R \alpha(v)$.*

Proof. In case of a collapsing rule, any instantiations of x appearing as depth-1 subterms of u are equal to v, and so they are replaced by $\phi(v)$ in $\alpha(u)$. Since constants in w are never replaced, $\alpha(u) \leftrightarrow_R \phi(v)$. Otherwise, if s is a depth-1 subterm of u or v that is an instantiation of a shared variable, then every depth-1 instance of s is replaced by $\phi(s)$ in $\alpha(u)$ and $\alpha(v)$. So, $\alpha(u) \leftrightarrow_R \alpha(v)$. ■

▶ **Proposition 14.** *Let R be a flat TRS. Let s and t be terms not equivalent to a constant and $\pi \in s \overset{*}{\leftrightarrow} t$ be a proof over R. Then, either there is a proof $\alpha(s) \overset{*}{\leftrightarrow}_{\widehat{E_R}} y$ for some variable y, or there is a proof $\alpha(s) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \alpha(t)$.*

Proof. We know that there is a proof, π_{1rr}, over $\widehat{E_R}$ with at most one root rewrite. If π_{1rr} has zero steps, then $\alpha(s) = \alpha(t)$, and so $\alpha(s) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \alpha(t)$. Assume that π_{1rr} has at least one step, and say that it has the form $s = s_0 \leftrightarrow_{\widehat{E_R}} \ldots \leftrightarrow_{\widehat{E_R}} s_k = t$ for some $k \geq 1$. We consider three cases: (i) π_{1rr} has no root rewrite; (ii) the only root rewrite in π_{1rr} uses a collapsing rule; and (iii) the only root rewrite in π_{1rr} does not use a collapsing rule.

In cases (i) and (iii), we know, by lemmas 12 and 13, that there is a proof $\alpha(s_i) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \alpha(s_{i+1})$ for $0 \leq i \leq k-1$. Therefore, there is a proof $\alpha(s) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \alpha(t)$.

In case (ii), let $w\sigma = s_j \leftrightarrow_{\widehat{E_R}} s_{j+1} = x\sigma$ be the instance of the collapsing rule, $w \to x$, for some $0 \leq j \leq k-1$. For $i < j$, we know that there is a proof $\alpha(s_i) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \alpha(s_{i+1})$. By Lemma 13, we know that $\alpha(s_j) \leftrightarrow_{\widehat{E_R}} \phi(s_{j+1})$, and so there is a proof $\alpha(s) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \phi(s_{j+1})$. Since the terms in π_{1rr} cannot be equivalent to a constant (since s, t are not equivalent to a constant), we know that $\phi(s_{j+1}) = x_{\overline{s_{j+1}}}$, and so the proof is complete ■

▶ **Remark 15.** *As mentioned above, for any term v not equivalent to a constant, $\phi(v)$ can be chosen so that it does not appear as a subterm of any finite number of terms. Therefore, $\phi(s_{j+1})$ can be chosen so that it does not appear as a subterm of $s_0, s_1, \ldots, s_k$.*

▶ **Proposition 16.** *Let R be a flat TRS, and let $\langle M, N\rangle$ be a minimal witness to non-$UN^=$ for R, with M, N not equivalent to a constant. Then either $\langle \alpha(M), y\rangle$ or $\langle \alpha(M), \alpha(N)\rangle$ is a witness for some variable, y.*

Proof. We know from Proposition 14 that either there is a proof $\alpha(M) \overset{*}{\leftrightarrow}_{\widehat{E_R}} y$ for some variable y, or there is a proof $\alpha(M) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \alpha(N)$. So, we need to show that (i) $\alpha(M)$, $\alpha(N)$, and y are normal forms, and that (ii) $\alpha(M) \neq y$ (whenever $\alpha(M) \overset{*}{\leftrightarrow}_{\widehat{E_R}} y$) and $\alpha(M) \neq \alpha(N)$.

For (i), we need to show that if s and t are depth-1 subterms of M (or N) that are not equivalent to constants, then $\phi(s) \neq \phi(t)$ whenever $s \neq t$. So, say that $s \neq t$. If $s \overset{*}{\leftrightarrow}_{\widehat{E_R}} t$, then $\langle s, t\rangle$ would violate the minimality of $\langle M, N\rangle$, since $|s| + |t| < |M| \leq |M| + |N|$. So, we know that s and t are not equivalent, and hence $\phi(s) \neq \phi(t)$. We know by Proposition 4 that $\alpha(M)$ and $\alpha(N)$ are normal forms, because the variables replacing subterms of M and N can be chosen so that they are fresh. Since variables are always normal forms, we know that $\alpha(M)$, $\alpha(N)$, and y are normal forms.

For (ii), if M is not a variable, then $\alpha(M)$ is not a variable, and hence $\alpha(M) \neq y$. If M is a variable, then, by Remark 15, we can choose y so that it does not appear as a subterm of M. So, $\alpha(M) = M \neq y$.

To see that $\alpha(M) \neq \alpha(N)$, we need to consider two cases. If $root(M) \neq root(N)$, then clearly $\alpha(M) \neq \alpha(N)$, since α does not affect the outermost function symbol. If $root(M) = root(N)$, then it must be the case that $M|_i \neq N|_i$ for some integer, i. In order for $\alpha(M) = \alpha(N)$ to be true, $M|_i$ and $N|_i$ must be replaced by the same variable. But this only happens when $M|_i$ and $N|_i$ are equivalent, and if $M|_i$ and $N|_i$ were equivalent, then (setting $M' = M|_i$ and $N' = N|_i$) $\langle M', N'\rangle$ would be a witness with $|M'| < |M|$ and $|N'| < |N|$. This would violate the minimality of $\langle M, N\rangle$, so $M|_i$ and $N|_i$ cannot be equivalent, and hence $M|_i$ and $N|_i$ must be replaced by distinct variables. Therefore, $\alpha(M) \neq \alpha(N)$. ■

3 Decidability for Flat and Shallow Rewrite Systems

▶ **Lemma 17.** *Let R be a flat TRS, and say that there is no constant $c \in \mathcal{F}_0$ and normal form $N' \in SubMinWit_R$ such that $c \overset{*}{\leftrightarrow}_{\widehat{E_R}} N'$ and $height(N') \geq |\mathcal{F}_0|$. Let $\langle M, N\rangle$ be a minimal witness to non-$UN^=$ for R. Then $height(\alpha(M)), height(\alpha(N)) \leq k = max\{1, |\mathcal{F}_0|\}$.*

Proof. We know that (i) all depth-1 subterms of $\alpha(M)$ and $\alpha(N)$ that are not equivalent to a constant are necessarily variables, and (ii) there is no constant $c \in \mathcal{F}_0$ and normal form $N' \in SubMinWit_R$ such that $c \overset{*}{\leftrightarrow}_{\widehat{E_R}} N'$ and $height(N') \geq |\mathcal{F}_0|$. Hence, the depth-1 subterms of $\alpha(M)$ and $\alpha(N)$ are either (i) variables or (ii) elements of $SubMinWit_R$ with

height strictly less than $|\mathcal{F}_0|$. This means that the heights of $\alpha(M)$ and $\alpha(N)$ are at most $max\{1, |\mathcal{F}_0|\}$. ■

▶ **Theorem 1.** *Let R be a flat TRS. If there is a witness to non-$UN^=$ for R, then there exists a witness, $\langle N_0, N_1\rangle$, with $height(N_0), height(N_1) \leq k = max\{1, |\mathcal{F}_0|\}$. Hence $UN^=$ is decidable for R.*

Proof. By Proposition 10, we know that there is either (i) *no* constant $c \in \mathcal{F}_0$ and normal form $N' \in SubMinWit_R$ such that $c \overset{*}{\leftrightarrow}_{\widehat{E_R}} N'$ and $height(N') \geq |\mathcal{F}_0|$, or (ii) a witness, $\langle N_0, N_1\rangle$ to non-$UN^=$ for R such that $height(N_0), height(N_1) \leq k$. Further, there is an effective procedure to decide if (i) or (ii) is the case.

If (ii) is the case, then we have our witness. So, assume that (i) is the case, and let $\langle M, N\rangle$ be a minimal witness to non-$UN^=$ for R. If M and N are equivalent to a constant, c, and $height(M)$, $height(N) < |\mathcal{F}_0|$, then we are done. So, we assume (without loss of generality) that M, N are not equivalent to a constant, and thus we can apply Proposition 14. Hence there is either a proof $\alpha(M) \overset{*}{\leftrightarrow}_{\widehat{E_R}} y$ for some variable y, or a proof $\alpha(M) \overset{*}{\leftrightarrow}_{\widehat{E_R}} \alpha(N)$. By Lemma 17, we know that $height(\alpha(M)), height(\alpha(N)) \leq k$. Hence, by Proposition 16, either $\langle \alpha(M), y\rangle$ or $\langle \alpha(M), \alpha(N)\rangle$ is a witness to non-$UN^=$ with $height(\alpha(M)), height(\alpha(N)), |y| \leq k$.

So, if there is a witness to non-$UN^=$ for R, then there is a witness, $\langle N_0, N_1\rangle$, with $height(N_0)$, $height(N_1) \leq k$. The following algorithm, on input R, determines if R is $UN^=$: Enumerate all ground normal forms over the signature of the rewrite system, i.e., consisting of constants and function symbols appearing in the finitely many rules of R, with height less than, or equal to, k; say they are $N_0, \ldots, N_n$. In [1], the authors show that the word problem is decidable for shallow TRS. So, for $0 \leq i < j \leq n$, check if $N_i \overset{*}{\leftrightarrow}_{\widehat{E_R}} N_j$. If $N_i \overset{*}{\leftrightarrow}_{\widehat{E_R}} N_j$ for some $0 \leq i < j \leq n$, then R is not $UN^=$; otherwise, R is $UN^=$. ■

Now that we have shown that $UN^=$ is decidable for flat rewrite systems, we extend this result to shallow rewrite systems. We do this by *flattening* a shallow rewrite system, i.e., transforming a shallow rewrite system into a flat one in a way that preserves $UN^=$.

▶ **Theorem 2.** *Let R be a shallow TRS. Then $UN^=$ is decidable for R.*

4 Undecidability of $UN^=$ for some Rewrite Systems

We show that $UN^=$ is undecidable for certain rewrite systems by showing that a decision procedure for $UN^=$ for these rewrite systems could be used to construct a decision procedure for the Post Correspondence Problem (PCP) [8]. As PCP is undecidable, so must $UN^=$ be for these rewrite systems. Note that, in this section, we sometimes use concatenation to denote the application of a unary function, i.e., $f(g(h(c)))$ could, for convenience, be denoted by $fgh(c)$. We consider rewrite systems with rules that have flat right-hand sides, and left-hand sides with height at most two. For each PCP instance, P, with tiles $\tau_1, \ldots, \tau_k$ (each tile is basically a pair of strings) and tile alphabet Γ, we construct a TRS, R_P, with flat right-hand sides, and left-hand sides with height at most two. Let T be the set of tiles, and say Γ_{bot} is the set of words appearing on the bottom of a tile, and Γ_{top} is the set of words appearing on the top of a tile in T. We construct the TRS as follows:

1. For each tile τ_i, $f(i(x), u(y), v(z)) \to f(x, y, z)$, where $u \in \Gamma_{top}$ is on the top of tile τ_i, and $v \in \Gamma_{bot}$ is on the bottom of τ_i.
2. For constants s and α, $f(s, s, s) \to \alpha$.
3. For each $a \in \Gamma$ and each tile τ_i, $f(i(x), a(y), a(y)) \to \beta$, where β is a constant.
4. $s \to s$, $f(x, y, z) \to f(x, y, z)$, $a(x) \to a(x)$, and $i(x) \to i(x)$ for every $a \in \Gamma$ and tile τ_i.

The rules from item (1) are used to construct the bulk of the proof. The rule from (3) allows you to reach the normal form β once a PCP instance has been constructed. Notice that α and β are the only two normal forms for R_P. The rules from item (4) do not play a non-trivial role in any proof—they exist simply to eliminate the possibility of there being more than two normal forms. Notice that $u, v \in \Gamma^+$ appear on the left-hand side of a rule. This means that the left-hand side can have height strictly greater than two. However, putting u, v on the left-hand side of a rule is just a convenience, as such a rule can be simulated by rules with flat right-hand sides, and left-hand sides with height at most two. For instance, let $u = \gamma_m \dots \gamma_1$ and $v = \delta_n \dots \delta_1$, for $\gamma_i, \delta_j \in \Gamma$ and $n \geq m$. In this case, the rule $f(i(x), u(y), v(z)) \to f(x, y, z)$ can be simulated by:

$$\begin{array}{rcc} f(i(x), y, \delta_n(z)) & \to & f^{(n-1)}(x, y, z) \\ & \vdots & \\ f^{(m+1)}(x, y, \delta_{m+1}(z)) & \to & f^{(m)}(x, y, z) \\ f^{(m)}(x, \gamma_m(y), \delta_m(z)) & \to & f^{(m-1)}(x, y, z) \\ & \vdots & \\ f^{(2)}(x, \gamma_2(y), \delta_2(z)) & \to & f^{(1)}(x, y, z) \\ f^{(1)}(x, \gamma_1(y), \delta_1(z)) & \to & f(x, y, z) \end{array}$$

We given an outline of the proof of correctness and omit details for lack of space.

▶ **Lemma 18.** *A minimal proof over R_P cannot contain a backward application of rule type* 1 *at the root position immediately followed by a forward application at the root position of rule type* 1*.*

▶ **Lemma 19.** *Let $\alpha \overset{*}{\leftrightarrow} \beta$ be a proof over R_P with minimal length. Then the proof must have the form $\alpha \leftrightarrow f(s, s, s) \overset{+}{\leftrightarrow} f(i(t'), a(t), a(t)) \leftrightarrow \beta$.*

▶ **Corollary 20.** *Let P be a PCP instance. If R_P is not* $\mathrm{UN}^{=}$*, then there is a solution to the PCP instance.*

It is straight-forward to show that if there is a solution to P, then R_P violates $\mathrm{UN}^{=}$. So, if P is an instance of PCP, then there is a solution to P if and only if R_P violates $\mathrm{UN}^{=}$. Since PCP is undecidable, we have the following theorem.

▶ **Theorem 3.** *$UN^{=}$ is undecidable for TRS with rules that have flat, linear right-hand sides and left-hand sides with height at most two.*

A slight modification of the rules can produce another result. Consider the following rule set:

1. For each tile τ_i, $f(i(x), u(y), v(z)) \leftarrow f(x, y, z)$, where $u \in \Gamma_{top}$ is on the top of tile τ_i, and $v \in \Gamma_{bot}$ is on the bottom of τ_i.
2. For constants s and α, $f(s, s, s) \to \alpha$.
3. For each $a \in \Gamma$ and each tile τ_i, $f(i(x), a(y), a(y)) \leftarrow \beta$.
4. $\beta \to \gamma$, where γ is a constant.
5. $s \to s$, $f(x, y, z) \to f(x, y, z)$, $a(x) \to a(x)$, and $i(x) \to i(x)$ for every $a \in \Gamma$ and tile τ_i.

Notice that now α and γ are the only two normal forms. Let $\alpha \overset{*}{\leftrightarrow} \gamma$ be a proof over R_P with minimal length. Then the proof must have the form $\alpha \leftrightarrow f(s, s, s) \overset{+}{\leftrightarrow} f(i(t'), a(t), a(t)) \leftrightarrow \beta \leftrightarrow \gamma$. So, we have the following corollary.

▶ **Corollary 21.** *$UN^{=}$ is undecidable for TRS with rules that have linear, flat left-hand sides, and right-hand sides with height at most two.*

5 Conclusion

The $UN^{=}$ property of TRSs is shown to be decidable for the shallow class and undecidable for the class of TRSs in which one side of the rule is allowed to be at most depth-two and the other side is flat and linear. Among the fundamental properties of TRSs only the word problem and the $UN^{=}$ property are now known to be decidable for the shallow class. An important direction for future research is to give a complete classification of the basic properties for all 15 classes obtained by combinations of linearity and depth restrictions on variables in each side of TRSs (see also [11] in this regard).

Acknowledgements We thank Ross Greenwood for a careful reading and for his comments and questions.

References

1 H. Comon, M. Haberstrau, and J. Jouannaud. Syntacticness, cycle-syntacticness, and shallow theories. *Inf. Comput.*, 111(1):154–191, 1994.

2 G. Godoy, A. Tiwari, and R. Verma. On the confluence of linear shallow rewrite systems. *Proceedings of the Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science*, 2607:85–96, 2003.

3 Guillem Godoy and Hugo Hernández. Undecidable properties of flat term rewrite systems. *Appl. Algebra Eng. Commun. Comput.*, 20(2):187–205, 2009.

4 Guillem Godoy and Sophie Tison. On the normalization and unique normalization properties of term rewrite systems. In *Proc. Conf. on Automated Deduction*, pages 247–262, 2007.

5 J.W. Klop. *Combinatory Reduction Systems*. PhD thesis, Mathematisch Centrum, Amsterdam, 1980.

6 J.W. Klop. Rewrite systems. In *Handbook of Logic in Computer Science*. Oxford, 1992.

7 Ichiro Mitsuhashi, Michio Oyamaguchi, and Florent Jacquemard. The confluence problem for flat TRSs. In *8th Artificial Intelligence and Symbolic Computation Conference*, pages 68–81, 2006.

8 M. Sipser. *Introduction to the Theory of Computation*. PWS Publishing Co., 1997.

9 Terese. *Term Rewriting Systems*. Cambridge University Press, Cambridge, 2003.

10 Rakesh Verma. Complexity of normal form problem and reductions for term rewriting problems. *Fundamenta Informaticae*, 92(1-2):145–168, 2009.

11 Rakesh M. Verma. New undecidability results for properties of term rewrite systems. In *Proc. (elec.) of 9th Workshop on Rule-based Programming (RULE)*, 2008.

12 R.M. Verma, M. Rusinowitch, and D. Lugiez. Algorithms and reductions for rewriting problems. *Fundamenta Informaticae*, 46(3):257–276, 2001. Also in Proc. of Int'l Conf. on Rewriting Techniques and Applications 1998.

13 J. Zinn and R. Verma. A polynomial-time algorithm for uniqueness of normal forms of linear, shallow rewrite systems. In *Proc. IEEE Conf. on Logic in Computer Science*, 2006. short presentation.

14 Julian Zinn. A polynomial algorithm for uniqueness of normal forms of linear, shallow term rewrite systems. Master's thesis, University of Houston, 2006.

Deterministic Black-Box Identity Testing π-Ordered Algebraic Branching Programs*

Maurice Jansen[1], Youming Qiao[2], and Jayalal Sarma M.N.[3]

1 **School of Informatics, The University of Edinburgh,** maurice.julien.jansen@gmail.com
2 **Institute for Theoretical Computer Science, Tsinghua University,** jimmyqiao86@gmail.com
3 **Department of Computer Science and Engineering, Indian Institute of Technology Madras,** jayalal.sarma@gmail.com

Abstract

In this paper we study algebraic branching programs (ABPs) with restrictions on the order and the number of reads of variables in the program. An ABP is given by a layered directed acyclic graph with source s and sink t, whose edges are labeled by variables taken from the set $\{x_1, x_2, \ldots, x_n\}$ or field constants. It computes the sum of weights of all paths from s to t, where the weight of a path is defined as the product of edge-labels on the path. Given a permutation π of the n variables, for a π-*ordered* ABP (π-OABP), for any directed path p from s to t, a variable can appear at most once on p, and the order in which variables appear on p must respect π. One can think of OABPs as being the arithmetic analogue of ordered binary decision diagrams (OBDDs). We say an ABP A is of read r, if any variable appears at most r times in A.

Our main result pertains to the polynomial identity testing problem, i.e. the problem of deciding whether a given n-variate polynomial is identical to the zero polynomial or not. We prove that over any field $\mathbb{F}$, and in the black-box model, i.e. given only query access to the polynomial, read r π-OABP computable polynomials can be tested in DTIME$[2^{O(r \log r \cdot \log^2 n \log\log n)}]$. In case $\mathbb{F}$ is a finite field, the above time bound holds provided the identity testing algorithm is allowed to make queries to extension fields of $\mathbb{F}$. To establish this result, we combine some basic tools from algebraic geometry with ideas from derandomization in the Boolean domain.

Our next set of results investigates the computational limitations of OABPs. It is shown that any OABP computing the determinant or permanent requires size $\Omega(2^n/n)$ and read $\Omega(2^n/n^2)$. We give a multilinear polynomial p in $2n+1$ variables over some specifically selected field $\mathbb{G}$, such that any OABP computing p must read some variable at least 2^n times. We prove a strict separation for the computational power of read $(r-1)$ and read r OABPs. Namely, we show that the elementary symmetric polynomial of degree r in n variables can be computed by a size $O(rn)$ read r OABP, but not by a read $(r-1)$ OABP, for any $0 < 2r-1 \leq n$. Finally, we give an example of a polynomial p and two variables orders $\pi \neq \pi'$, such that p can be computed by a read-once π-OABP, but where any π'-OABP computing p must read some variable at least 2^n times.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.296

1 Introduction

The polynomial identity testing problem (PIT) is the question of deciding, given an arithmetic circuit C with input variables $x_1, x_2 \ldots x_n$ over some field $\mathbb{F}$, whether the polynomial computed

* This work was supported in part by the National Natural Science Foundation of China Grant 60553001, 61073174, 61033001 and the National Basic Research Program of China Grant 2007CB807900, 2007CB807901.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 296–307

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

by C is identical to the zero polynomial in the ring $\mathbb{F}[x_1, x_2, \ldots x_n]$. Efficient algorithms for PIT are important both in theory and in practice. Randomized algorithms were given independently by Schwartz [17] and Zippel [22].

Finding *deterministic* algorithms for PIT plays a crucial role in computational complexity theory. Kabanets and Implagliazzo [11] showed that giving a deterministic subexponential time algorithm for PIT implies that either NEXP $\not\subseteq$ P/*poly*, or that the permanent has no *poly*-size arithmetic circuits. Agrawal [1] showed that giving a deterministic *black-box* algorithm for PIT yields an explicit multilinear polynomial that has no subexponential size arithmetic circuits. In [1] a program was outlined explaining how making progress towards the latter kind of algorithm for PIT has the potential of resolving Valiant's Hypothesis, which states that the algebraic complexity classes VP and VNP are distinct. For optimists certainly, the situation is tantalizing, as Agrawal and Vinay [2] showed that the black-box derandomization of PIT for only depth-4 circuits would yield a nearly complete derandomization for general arithmetic circuits. Recent progress on the PIT problem has been impressive. See [16] for a recent survey.

In this paper, we contribute to the above mentioned lower bounds program by considering black-box identity testing *ordered algebraic branching programs* (OABPs), which where introduced in [8]. Algebraic branching programs have computational power somewhere in between arithmetic formulas and circuits. Namely, they can efficiently simulate formulas via a construction by Valiant [21]. Furthermore, their computational power is easily seen to be equivalent to that of skew circuits. For skew circuits, which were introduced by Toda [20], multiplication gates are restricted to have one of their inputs to be a variable or field constant. OABPs can be thought of as being the arithmetic analogue of *ordered binary decision diagrams* (OBDDs), which were introduced by Bryant [4].

Some polynomials can be succinctly represented in the OABP model. For example, we show that the elementary symmetric polynomial of degree k in n variables can be elegantly described by a grid shaped OABP of size $O(kn)$. This can be done for any desired variable order π, and shows small OABPs have some real computing power. We think the OABP model has practical merit for polynomial representation, and being an analogue of the OBDD it should be properly investigated. As our lower bounds show, a succinct OABP-representation is not available for every polynomial. The situation is similar to what is well-known for OBDDs. In practice this may be outweighed by the fact that PIT can be solved efficiently for OABPs. Part of the popularity of OBDDs can be explained by the fact that identity testing (and hence equivalence testing) can be done efficiently for the model, as e.g. Raz and Shpilka [14] remarked.

In [14] a polynomial-time non-black-box algorithm was given for identity testing non-commutative formulas, and more generally non-commutative ABPs. Identity testing OABPs reduces to PIT for non-commutative ABPs, and hence can be done *non-black-box* in polynomial time. Namely, if we take an OABP A computing some polynomial f over commuting variables, and if we let f' be the evaluation of A, where we restrict the variables to be non-commuting, then it can be observed that $f \equiv 0 \Leftrightarrow f' \equiv 0$. Giving a *black-box* algorithm for testing non-commutative formulas and ABPs is currently a major open problem. Our main result implies that for any variable order π, we have a DTIME$[2^{O(polylog(n))}]$ black-box algorithm for testing OABPs with order π that have $polylog(n)$ many reads.

Let us mention the connection of our work to the problem of identity testing multilinear formulas raised by Raz [13]. Our results can be applied to black-box identity testing "ordered multilinear formulas" with few reads (say $polylog(n)$). The latter can be defined for any given variable order π, by requiring that for each multiplication gate $g = g_1 \times g_2$ in the

formula, variables in the subformula rooted at g_1 should either all be smaller or all be larger w.r.t. π than variables in the subformula rooted at g_2. By applying the construction of [21], judiciously to keep the order, a formula of this kind can be simulated by a π-OABP. This then gives another important special case of PIT for multilinear formulas for which a black-box algorithm is known. The other case being sum-of-k read-once formulas, which we elaborate on next.

Any arithmetic *read-once formula* (ROF) can be simulated by an OABP, since the construction of [21] mentioned before preserves the RO-property. Black-box Identity testing sum-of-k ROFs was studied in [19], and this was subsequently generalized to the sum-of-k read-once ABPs in [10]. These results suggest the difficulty of making generalizations in this area to models beyond read-once. For example, by [10] we have an $n^{O(\log n)}$ black-box test for sum-of-two read-once ABPs, but for testing a single read-twice ABP, currently nothing is known beyond brute-force methods. Our result is significant, in that the techniques apply to a model where the multiple reads take place within one *monolithic* ABP. This opens up a new thread of progress in the direction of identity testing unrestricted ABPs. We refer to [9] for a direct connection between this, and proving lower bounds for the *determinantal complexity* of explicit polynomials. The latter is what the separation of VP and VNP requires.

Another point of significance pertains to the techniques we use (on which we will elaborate more below). To obtain the main result, we combine basic tools from algebraic geometry with ideas from derandomization in the Boolean world (specifically, the pseudorandom construction of Impagliazzo, Nisan and Wigderson [7] for network algorithms). As far as we now, this kind of use of basic algebraic geometry is new to the PIT area. We hope our work stimulates more research in this direction.

1.1 Techniques

Towards the identity testing algorithm, first we show that, without increasing the number of reads, any π-OABP can be made π-*oblivious*. For the latter, all variables in a layer must be identical, and all occurences of a variable x_i appear in the same layer. Hence there is some variable order $x_{i_1}, x_{i_2}, \ldots, x_{i_n}$ in which the layers appear (possibly interleaved by constant layers), when going from the source to the sink. The next step is to construct a generator $\mathcal{G}(z)$ for π-oblivious ABPs. This is a mapping $\mathbb{F}^\ell \to \mathbb{F}^n$ such that for any $f \in \mathbb{F}[x_1, \ldots, x_n]$ computed by a π-oblivious ABP, $f \equiv 0 \Leftrightarrow f(\mathcal{G}) \equiv 0$. From this, one obtains an efficient black-box test, if the number of z-variables ℓ and the degree of $\mathcal{G}$ is "small".

For illustrative purposes, let us consider an π-oblivous ABP A with variable order $x_1, x_2, \ldots, x_{2n}$ of small width w, rather than small number of reads, and suppose it computes $f \not\equiv 0$. In order to achieve $\ell = O(w \log n)$, we cut A in the middle layer. This gives a decomposition (say) $f = \sum_{i\in[w]} g_i(x_1, \ldots, x_n)h_i(x_{n+1}, \ldots, x_{2n})$. Then we want $f(\mathcal{G}) = \sum_{i\in[w]} g_i(\mathcal{G}_1, \ldots, \mathcal{G}_n)h_i(\mathcal{G}_{n+1}, \ldots, \mathcal{G}_{2n}) \not\equiv 0$. We would like to use recursion on the g_is and h_is, but in order to get ℓ small, this means $\mathcal{G}^u := (\mathcal{G}_1, \ldots, \mathcal{G}_n)$ and $\mathcal{G}^d := (\mathcal{G}_{n+1}, \ldots, \mathcal{G}_{2n})$ will share most of the variables. Consequently, cancelations might occur and may result in $f(\mathcal{G}) \equiv 0$. However, we do know that $\{g_i(\mathcal{G}^u)\}_{i\in[w]}$ must "communicate" through a small dimensional space $\mathbb{F}^w$. This allows one to take $\mathcal{G}^d$ identical to $\mathcal{G}^u$, except for an additional component to the input that inflates the dimension of any non-empty finite union of *affine varieties*[1], given by the preimage of a single point in $\mathbb{F}^w$. More or less, $\mathcal{G}(z, z')$ will look

[1] Keeping with the terminology in [6], an *algebraic set* is the set of common zeroes of a list of polynomials. Affine varieties are algebraic sets, which are *irreducible* in the *Zariski-topology*.

like $\mathcal{G}^u(z); \mathcal{G}^d(z, z')$, with $\mathcal{G}^d(z, z') = \mathcal{G}^u(z + T(z'))$, where T is a mapping of $O(w)$ many variables that contains any w-dimensional coordinate subspace. Doing so, we only add $O(w)$ many variables per inductive step. This mirrors the pseudorandom generator construction of [7] mentioned before. To make an analogy, $z + T(z')$ can be thought of as similar to taking a vertex (we pick z) and adjacent edge (we move by $T(z)$) on an expander graph.

Necessarily, our final construction will be more complicated than the above sketch, since we assume a bound on the number of reads instead of the width. This will be dealt with by taking a partial derivatives w.r.t. a centrally local variable x_k in the ABP. Taking the derivative w.r.t. x_k has the net effect of cutting down the width of the x_k-layer of A.

2 Preliminaries

For a natural number n, we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. For an n-tuple $a = (a_1, a_2, \ldots, a_n)$ and m-tuple $b = (b_1, b_2, \ldots, b_m)$, we denote $(a_1, a_2, \ldots, a_n, b_1, b_2, \ldots, b_m)$ by $a\#b$. Let $X = \{x_1, x_2, \ldots, x_n\}$ be a set of variables and let $\mathbb{F}$ be a field. For a polynomial $f \in R := \mathbb{F}[X]$, if it is identical to the zero polynomial of the ring R, we write $f \equiv 0$. If the degree of any variable of f is bounded by one, f is said to be *multilinear* (even if f has a constant term). We say f *depends on* x_i, if the formal partial derivative $\partial f/\partial x_i \not\equiv 0$. $Var(f)$ denotes the sets of variables f depends on. For a set of polynomial $f_1, \ldots, f_m \in \mathbb{F}[X]$, we say that they are *independent* if for all $a \in \mathbb{F}^m$ with $a \neq \overline{0}$, $\sum_{i\in[m]} a_i f_i \not\equiv 0$. We use the notation $f_{|x_i=\alpha}$ to denote substitution of x_i with $\alpha \in \mathbb{F}$.

We import the following definition and subsequent notations from [10]. An algebraic branching program (ABP) is a 4-tuple $A = (G, w, s, t)$, where $G = (V, E)$ is an edge-labeled directed acyclic graph for which the vertex set V can be partitioned into levels $L_0, L_1, \ldots, L_d$, where $L_0 = s$ and $L_d = t$. Vertices s and t are called the source and sink of A, respectively. Edges may only go between consecutive levels L_i and L_{i+1}. The subgraph induced by $L_i \cup L_{i+1}$ is called a *layer*. The label function $w : E \to X \cup \mathbb{F}$ assigns variables or field constants to the edges of G. For a path p in G, we extend the weight function by $w(p) = \prod_{e\in p} w(e)$. Let $P_{i,j}$ denote the collection of all paths p from i to j in G. The program A computes the polynomial $\sum_{p\in P_{s,t}} w(p)$. The size of A is taken to be $|V|$, and the read of A is the maximum of $|w^{-1}(x_i)|$, over all x_i's. The depth of A equals d, and the width of A equal $\max_i |L_i|$.

Algebraic branching programs were first introduced by Nisan [12]. Our definition differs in the respect that [12] requires edge labels to be linear forms. We remark that the read of an ABP always refers to *global read*, i.e. it bounds the total number of times a variable x_i can be reads in the entire ABP. With some abuse, an ABP A is called a read r ABP, if its read is bounded by r. We also denote this by saying that A is a R_r-ABP. A polynomial $f \in \mathbb{F}[X]$ is called a R_r-ABP-polynomial if there exists a R_r-ABP computing f. We use the following notation: for an arc $e = (v, w)$ in ABP A, $begin(e) = v$ and $end(e) = w$. We let $source(A)$ and $sink(A)$ stand for the source and sink of A. For any nodes v, w in A, we denote the subprogram with source v and sink w by $A_{v,w}$. We use $\widehat{A}$ to denote the polynomial computed by A, and in particular, $\widehat{A_{v,w}}$ is the polynomial computed by the subprogram $A_{v,w}$. A *layer* of an ABP A is the subgraph induced by two consecutive levels L_i and L_{i+1} in A.

▶ **Definition 1.** Let π be a permutation of $[n]$. An ABP A is π-ordered, if on every directed path p in A, if a variable x_i appears before x_j on p, then $\pi(i) < \pi(j)$. For an ABP A we say it is ordered if it is π-ordered w.r.t. some permutation π.

For a π-ordered ABP (π-OABP) variables appear (with possibly omissions) on any path from source to sink in the order $x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)}, \ldots, x_{\pi^{-1}(n)}$. We will speak of the latter

sequence as the *variable order* of A. To stress, for a π-OABP, for any path p, a variable x_i appears at most once on p. Hence, if a π-OABP is read r, each variable x_i can appear at most r times in the ABP, and each occurrence must be on a different path from the source to the sink. Note that the output of a π-OABP must be a multilinear polynomial. Ordered algebraic branching programs where first studied in [8], but with respect to the homogeneous ABP definition of [12]. There the ordering condition states that on any path p, for any edge e_1 appearing before e_2 on p, if e_1 is labeled by $\sum_{i\in[n]} a_i x_i$, and e_2 is labeled by $\sum_{i\in[n]} b_i x_i$, then all variables in $\{x_i : a_i \neq 0\}$ appear before all variables in $\{x_i : b_i \neq 0\}$ in the variable order. The usual "homogenization trick" of splitting nodes into parts computing homogeneous components can be used to convert any OABP to the model of [8] (one also needs to collapse circuitry going over constant wires). This outlines a proof of the second part of the following lemma (the first part being obvious):

▶ **Lemma 2.** *For any permutation π of $[n]$ we have the following:*

1. *A homogeneous π-ordered ABP of size s with linear forms as edge labels can be converted into an equivalent π-OABP with weight function $w : E \to X \cup \mathbb{F}$ of size $O(ns)$.*
2. *For any π-OABP of size s computing a homogeneous polynomial of degree d, there exists an equivalent homogeneous π-ordered ABP of size $O(sd)$ with linear forms as edge labels.*

An ABP is called *oblivious*, if for any layer all variables are the same. We call a layer an x-layer, if x labels some of the edges in that layer, for $x \in X$. Layers with variables are called *variable layers*. Layers without variables are called *constant layers*. We say an ABP is *π-oblivous*, if it is oblivious, *and* for each variable x_i there is at most one x_i-layer, *and* the layers appear in the order $x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)}, \ldots, x_{\pi^{-1}(n)}$ (with possible omissions) in the ABP.

To emphasize, for a read r π-oblivious ABP we have at most n layers where variables are read. These layers appear in the variable order when going from the source to the sink, and can be interleaved with constant labeled layers. Then for a variable layer w.r.t. a variable x, we have at most r occurences of x on an edge in this layer. Any remaining edges in the layer must be labeled by constants. The proof of the following lemma follows by some straightforward circuit manipulations, and it will appear in the full version of the paper. Note the lemma preserves read.

▶ **Lemma 3.** *For any permutation π of $[n]$, given a π-OABP A over n variables of size s and read r, there is an equivalent π-oblivious ABP B of size $O(sn)$, width $\leq 2s$, read r.*

Any subset $X \subseteq \mathbb{F}^n$ which is the set of simultaneous zeroes of a set of polynomials $f_1, \ldots, f_t \in \mathbb{F}[x_1, \ldots, x_n]$ is called an *algebraic set*. For basic definitions we refer to [5, 6]. If X and Y are algebraic sets in $\mathbb{F}^n$, we denote by $X+Y$ the subset $\{x+y \in F^n : x \in X, y \in Y\}$. Note that $X+Y$ may not be an algebraic set. We denote by $\overline{X+Y}$ the closure of $X+Y$ in the Zariski-topology. We need the following two lemmas:

▶ **Lemma 4.** *Let $X \subset \mathbb{F}^n$ be an algebraic set of dimension $0 \leq r < n$. Then for some $(n-r)$-dimensional coordinate subspace $C \subset \mathbb{F}^n$, $\overline{X+C} = \mathbb{F}^n$.*

Proof. For a coordinate subspace C denote the canonical projection to C by π_C. Consider $K = \{0\}^r \times \mathbb{F}^{n-r}$ and $L = \mathbb{F}^r$, which we think of as the complement of K corresponding to the first r coordinates. We have the following two properties: 1) The set $X + K$ equals $\pi_L(X) \times \mathbb{F}^{n-r}$, and 2) $\overline{\pi_L(X) \times \mathbb{F}^{n-r}} = \overline{\pi_L(X)} \times \mathbb{F}^{n-r}$.

By this, $\dim \overline{X+K} = n - r + \dim \overline{\pi_L(X)}$. More generally, it can be seen (by applying isomorphisms to $\mathbb{F}^n$, where we permute the indices), that for any $(n-r)$ -dimensional coordinate subspace C with r-dimensional complement D, $\dim \overline{X+C} = n - r + \dim \overline{\pi_D(X)}$.

Hence the lemma follows from the fact that for any r-dimensional affine variety there exists a projection τ to some r-dimensional coordinate subspace E such that $\tau(X)$ is dense in E, i.e. $\dim \overline{\pi_D(X)} = r$. For a proof of the latter see [5], p480. ◀

▶ **Lemma 5** (Lemma 2.1 in [3]). *Let $f \in \mathbb{F}[X]$ be a nonzero polynomial such that the degree of f in x_i is bounded by r_i, and let $S_i \subseteq \mathbb{F}$ be of size at least $r_i + 1$, for all $i \in [n]$. Then there exists $(s_1, s_2, \ldots, s_n) \in S_1 \times S_2 \times \ldots \times S_n$ with $f(s_1, s_2, \ldots, s_n) \neq 0$.*

The following lemma gives us a decomposition satisfying some useful independence properties.

▶ **Lemma 6.** *Let $k \geq 1$, and let A be an oblivious ABP of width w with source s and sink t having variable order $x_1, x_2, .., x_{2n}$. Suppose $\widehat{A} \not\equiv 0$. Then we can write for some $w' \leq w$, $f = \sum_{i \in [w']} f_i g_i$, where*

1. *$\{f_1, f_2, \ldots, f_{w'}\} \subseteq \mathbb{F}[x_1, x_2, \ldots, x_n]$ and $\{g_1, g_2, \ldots, g_{w'}\} \subseteq \mathbb{F}[x_{n+1}, x_{n+2}, \ldots, x_{2n}]$ are both independent sets of polynomials.*
2. *$\forall a \in \mathbb{F}^{w'}$, $\sum_{i \in [w']} a_i f_i$ can be computed by an oblivious ABP of width w with variable order $x_1, x_2, .., x_n$.*
3. *$\forall a \in \mathbb{F}^{w'}$, $\sum_{i \in [w']} a_i g_i$ can be computed by an oblivious ABP of width w with variable order $x_{n+1}, x_{n+2}, \ldots, x_{2n}$.*

Proof. Let V be the set of variables used in A. Pick an arbitrary level L of nodes $v_1, v_2, \ldots, v_w$ such that $V \cap \{x_1, x_2, \ldots, x_n\}$ appear on edges in layers before L, and $V \cap \{x_n, x_{n+1}, \ldots, x_{2n}\}$ appear on edges in layers after L. For $i \in [w]$, let $f_i = \widehat{A_{s,v_i}}$ and $g_i = \widehat{A_{v_i,t}}$. We proceed in two phases. First we arrange for a decomposition where the f_is are independent. Then we will deal with the g_is.

Wlog. assume that $f_1, \ldots, f_k$ is a maximum size independent set of polynomials. Since $f \not\equiv 0$, we know that not all $f_i \equiv 0$. So $k \geq 1$. For $j > 0$, any f_{k+j} can be written as a linear combination of $f_1, \ldots, f_k$. Let A' be an equivalent ABP obtained from A as follows. First, A' is just as A from the source up to the level L, except that we drop $v_{k+1}, \ldots v_w$ from L. Let us use L' to denote the modified level L. L' is followed by a constant layer, where $f_1, \ldots, f_w$ are computed (relative to s). After this we attach all the levels of A, just as they followed L in A. We have that $f = \sum_{i \in [k]} f_i g'_i$, where $f_i = \widehat{A'_{s,v_i}}$ and $g'_i = \widehat{A'_{v_i,t}}$. The f_is satisfy the first two conditions of the lemma. The g'_is are in $\mathbb{F}[x_{n+1}, x_{n+2}, \ldots, x_{2n}]$. This completes the first phase.

For the next phase, wlog. assume that $g'_1, \ldots, g'_l$ is a maximum size independent set. Say these correspond to nodes $w_1, \ldots, w_l$, respectively. That is, $\widehat{A'_{w_i,t}} = g'_i$. Since $f \not\equiv 0$, we know that $l \geq 1$. Symmetrically to the first phase, but now going in the direction from sink to source, we modify A' into an equivalent ABP A''. A'' is the same as A' from the sink back to the level L', except that we drop nodes other than $w_1, \ldots, w_l$ from L'. Above this is a constant level, where we compute $g'_1, \ldots, g'_k$ (relative to the sink). Above this we attach all level from A', just as they appear from s to L' in A'. We now have arranged that $f = \sum_{i \in [l]} f''_i g'_i$, where $f''_i = \widehat{A''_{s,w_i}}$ and $g'_i = A''_{w_i,t}$, for $i \in [l]$. Observe that for each $i \in [l]$, $f''_i = f'_i + \text{Linear}(f_{l+1}, \ldots, f_k)$. Hence $\{f''_1, \ldots, f''_l\}$ is an independent set of polynomials. All required properties of the lemma are now clearly satisfied. ◀

▶ **Corollary 7.** *Let $k \geq 1, n \geq 3$ and let $1 < i < n$. Let A be a read r oblivious ABP, with source s and sink t having variable order $x_1, x_2, \ldots, x_{i-1}, x_i, x_{i+1}, \ldots, x_n$. We use y as alias for x_i. Let $f = \partial \widehat{A} / \partial y$. Suppose $\widehat{A}$ depends on y, that is $f \not\equiv 0$. Then we can write for some $r' \leq r$, $f = \sum_{i \in [r']} p_i q_i$, where*

1. $\{p_1, p_2, \ldots, p_{r'}\} \subseteq \mathbb{F}[x_1, x_2, \ldots, x_{i-1}]$ *and* $\{q_1, q_2, \ldots, q_{r'}\} \subseteq \mathbb{F}[x_{i+1}, x_{i+2}, \ldots, x_n]$ *are both independent sets of polynomials.*
2. $\forall a \in \mathbb{F}^{r'}$, $\sum_{i\in[r']} a_i p_i$ *can be computed by a read* r *oblivious ABP with variable order* $x_1, x_2, .., x_{i-1}$.
3. $\forall a \in \mathbb{F}^{r'}$, $\sum_{i\in[r']} a_i q_i$ *can be computed by a read* r *oblivious ABP with variable order* $x_{i+1}, x_{i+2}, \ldots, x_n$.

Proof. Corollary 7 is now proved as follows. We make changes to A by modifying the edges in the y-layer as follows: for a variable edge (labeled with y), label it with 1. For a constant edge, remove it. The resulting ABP A' computes f. Then in the proof of Lemma 6, take the level L to be the starting level of the original y-layer. As $|L|$ is bounded by the number of y-variables in the y-layer of A, we are done. ◀

3 A Generator for π-Oblivious ABPs

We assume $|\mathbb{F}|$ is large enough. The explicit requirement on $|\mathbb{F}|$ will become clear after the description of the generator. For now, let us fix $S = \{\alpha_1, \ldots, , \alpha_N\} \subseteq \mathbb{F}$, for some N, and let $S_m = \{\alpha_1, \ldots, \alpha_m\}$, for $1 \leq m \leq N$. Let $Z = \{z_1, z_2, \ldots\}$, $Y = \{y_1, y_2, \ldots\}$, $U = \{u_1, u_2, \ldots\}$ and $V = \{v_1, v_2, \ldots\}$ be sets of variables. For $k \geq 1$, we use Z_k to denote the k-tuple of variables $(z_1, z_2, \ldots, z_k)$, similarly for Y_k, U_k and V_k. Define the function ℓ on natural numbers by $\ell(k, r) = 2rk + 1$. Abusing notation, we write $(Z_{\ell(k,r)}, U_k, V_k)$ to denote the tuple $Z_{\ell(k,r)} \# U_k \# V_k$.

For every $k \geq 0$, $r \geq 1$ and a variable w, let $H^{k,r}(w) = (H_1^{k,r}(w), H_2^{k,r}(w), \ldots H_{\ell(k,r)+2k}^{k,r}(w))$, where for each $i \in [\ell(k, r) + 2k]$, $H_i^{k,r}$ is the ith Lagrange interpolation polynomial on the set $S_{\ell(k,r)+2k}$. $H_i^{k,r}$ is a univariate polynomial in w of degree $\ell(k, r) + 2k - 1$, satisfying that $\forall \alpha_j \in S_{\ell(k,r)+2k}$, $H_i^{k,r}(\alpha_j) = 1$ if $i = j$ and 0 otherwise. For $k \geq 1$, and two variables u and v, let $E^k(u, v) = (u \cdot L_1^k(v), \ldots, u \cdot L_{2^k}^k(v))$, in which L_i^k is the ith Lagrange interpolation polynomial on the set S_{2^k}.

For $k \geq 0$ and $r \geq 1$, we define the polynomial mapping $F^{k,r}(Z_{\ell(k,r)}, U_k, V_k) : \mathbb{F}^{\ell(k,r)+2k} \to \mathbb{F}^{2^k}$ inductively as follows:

1. $F^{0,r}(z_1) = z_1$, and
2. For clarity we use $y_1, y_2, \ldots, y_{2r}$ as aliases for the variables $z_{\ell(k,r)+1}, z_{\ell(k,r)+2}, \ldots, z_{\ell(k,r)+2r}$, respectively. We take $F^{k+1,r}(Z_{\ell(k,r)}, Y_{2r}, U_{k+1}, V_{k+1})$ to be equal to the following 2^{k+1}-tuple of polynomials:

$$E^{k+1}(u_{k+1}, v_{k+1}) + \left[F^{k,r}(Z_{\ell(k,r)}, U_k, V_k) \# F^{k,r}\left((Z_{\ell(k,r)}, U_k, V_k) + T^{k,r}(Y_{2r})\right)\right],$$

where $T^{k,r} : \mathbb{F}^{2r} \to \mathbb{F}^{\ell(k,r)+2k}$ is defined by $T^{k,r}(Y_{2r}) = \sum_{i\in[r]} y_i \cdot H^{k,r}(y_{r+i})$.

From the construction we can see that in order to accommodate for S, $|\mathbb{F}|$ should be no less than $\max(\ell(k, r)+2k, 2^k)$. Note that the image of $T^{k,r}$ contains any r-dimensional coordinate subspace of $\mathbb{F}^{\ell(k,r)+2k}$. Namely, for $i \in [r]$, by choosing $y_{r+i} = \alpha_j$, the corresponding vector of $y_i H^{k,r}(y_{r+i})$ becomes $y_i e_j$, where e_j is the jth standard basis vector of $\mathbb{F}^{\ell(k,r)+2k}$. Thus by choosing different α's for the y_{r+i}'s, we can form any r-dimensional coordinate subspace in the image. The term E^{k+1} is there to deal with bounded read, e.g. it would not be needed if we want to have a generator for small width π-oblivious ABPs. Ignoring this term, the generator mimics the construction of [7]. Intuitively, the dimension expanding properties of $T^{k,r}$ will yield that the two sides of the generator appear to be behaving "independently enough", yielding the desired non-cancelation property.

3.1 Properties of the Generator

Let us compute $F^{1,r}$ to get a sense and for later use. We obtain

$$\begin{aligned} F^{1,r} &= E^1(u_1, v_1) + F^{0,r}(z_1) \# F^{0,r}(z_1 + (z_{1+1} + \cdots + z_{1+r})) \\ &= (u_1 L_1^1(v_1) + z_1, u_1 L_2^1(v_1) + z_1 + \cdots + z_{1+r}). \end{aligned}$$

Note that $z_{2+r}, \ldots, z_{1+2r}$ are not used in the Lagrange interpolation in the $T^{0,r}$ part. By a straightforward induction, one can prove the following bound for the individual degree of a variable in $F^{k,r}$.

▶ Proposition 1. $\forall k \geq 2$ and $r \geq 1$, the individual degree of any variable in any component of $F^{k,r}$ is at most $\prod_{j \in [k-1]} (\ell(j,r) + 2j)(\ell(j,r) + 2j - 1))$.

The following theorem shows that the generator $F^{k,r}$ works for the class $\mathcal{C}$ of polynomials computed by read r π-oblivious ABPs, where there is one single fixed order π of the variables for the entire class $\mathcal{C}$. Wlog. the order is assumed to be $x_1, x_2, \ldots$. A generator for any other fixed order, is obtained by permuting the components of the output of the generator in the appropriate way. To make the algebraic geometry go through in the proof, we will assume that $\mathbb{F}$ is algebraically closed. We will remove this requirement subsequently with Corollary 9.

▶ **Theorem 8.** *Let $\mathbb{F}$ be an algebraically closed field. Let $k \geq 0$, and let A be a π-oblivious ABP of read $r \geq 1$ with variable order $x_1, x_2, \ldots, x_{2^k}$. Suppose A computes f, then $f \equiv 0 \iff f(F^{k,r}) \equiv 0$.*

Proof. The "⇒"-direction is trivial, so it suffices to show that if $f \not\equiv 0$, then $f(F^{k,r}) \not\equiv 0$. We prove this by induction on k. For $k = 0$ it is obvious. For $k = 1$, we know there exists (a, b) such that $f(a, b) \neq 0$. Recall $F^{1,r} = (u_1 L_1^1(v_1) + z_1, u_1 L_2^1(v_1) + z_1 + \cdots + z_{1+r})$. Then setting c to be the assignment of (Z_{2r+1}, u_1, v_1) as $z_1 = a$, $z_2 = b - a$ and other variables to 0, would give $f(F^{1,r}) = f(a, b) \neq 0$. So $f(F^{1,r}) \not\equiv 0$.

Now let $k \geq 1$. For the induction step from k to $k+1$, we need to prove that $F^{k+1,r}$ works for an oblivious read r ABP polynomial f with variables $x_1, \ldots, x_{2^{k+1}}$. We use X as an alias for x_{2^k}, and Λ as an alias for α_{2^k}. Let $g = \partial f / \partial X$, and note that $f = g \cdot X + f \mid_{X=0}$, since f is multilinear. Wlog. we can assume that f depends on X. Namely, since f is multilinear, if f does not depend on any variable, i.e. $\forall i, \partial f / \partial x_i \equiv 0$, then $f \in \mathbb{F}$ (even if $char(\mathbb{F}) > 0$). Clearly the theorem holds in this case. Otherwise, the rest of the proof goes through mutatis mutandis by selecting X to be the median variable (w.r.t. the variable order $x_1, x_2, \ldots$) of variables that f depends on. Thus $g \not\equiv 0$. We claim that the following holds:

▶ Claim 1. $h := g(F^{k+1,r}) \mid_{v_{k+1}=\Lambda} \not\equiv 0$

Before proving Claim 1, let us show that this is sufficient to complete the proof of Theorem 8. We will prove Claim 1 in the next subsection. Consider $f(F^{k+1,r}) \mid_{v_{k+1}=\Lambda}$. It is equal to the following:

$$\begin{aligned} h \cdot \left(F_{2^k}^{k+1,r} \mid_{v_{k+1}=\Lambda}\right) + \left((f \mid_{X=0})(F^{k+1,r})\right) \mid_{v_{k+1}=\Lambda} &= \\ h \cdot \left((E_{2^k}^{k+1} + P(Z_{\ell(k,r)}, U_k, V_k)) \mid_{v_{k+1}=\Lambda}\right) + \left((f \mid_{X=0})(F^{k+1,r})\right) \mid_{v_{k+1}=\Lambda} &= \\ h \cdot (u_{k+1} + P(Z_{\ell(k,r)}, U_k, V_k)) + \left((f \mid_{X=0})(F^{k+1,r})\right) \mid_{v_{k+1}=\Lambda}, & \end{aligned}$$

for some polynomial P in variables $(Z_{\ell(k,r)}, U_k, V_k)$. Observe that $\left((f \mid_{X=0})(F^{k+1,r})\right) \mid_{v_{k+1}=\Lambda}$ does not contain the variable u_{k+1}. The same holds for $P(Z_{\ell(k,r)}, U_k, V_k))$. Hence $h \cdot u_{k+1}$ cannot be canceled, and therefore $f(F^{k+1,r}) \mid_{v_{k+1}=\Lambda} \not\equiv 0$. This implies $f(F^{k+1,r}) \not\equiv 0$. ◀

▶ **Corollary 9.** *Let $\mathbb{F}$ be any field. Let $k \geq 0$, and let A be a π-oblivious ABP over $\mathbb{F}$ of read $r \geq 1$ with variable order $x_1, x_2, .., x_{2^k}$. Suppose A computes the polynomial $f \in \mathbb{F}[x_1, x_2, .., x_{2^k}]$. Then in the construction of $F^{k,r}$ selecting any set S of size $\max(\ell(k,r) + 2k, 2^k)$ contained in $\mathbb{F}$ (or an arbitrary field extension $\mathbb{G}$ of $\mathbb{F}$, if $\mathbb{F}$ is not large enough) yields that $f \equiv 0 \iff f(F^{k,r}) \equiv 0$,*

Proof. First consider the case when $\text{char}(\mathbb{F}) = 0$. In this case we take $S = \{0, 1, 2, \dots\}$. Let $\overline{\mathbb{F}}$ be the algebraic closure of $\mathbb{F}$. Interpreting A as an ABP over $\bar{\mathbb{F}}$, we can apply Theorem 8 to conclude $f \equiv 0 \iff f(F^{k,r}) \equiv 0$. All coefficients of $F^{k,r}$ are rational numbers and thus lie inside $\mathbb{F}$. Hence the property $f \equiv 0 \iff f(F^{k,r}) \equiv 0$ also holds when considering we work over $\mathbb{F}$.

In case $\text{char}(\mathbb{F}) > 0$, if $|\mathbb{F}|$ is not large enough, by allowing ourselves to use elements from the extension $\mathbb{G}$, we can still get the required S. Then similarly as above, by considering the algebraic closure of $\mathbb{G}$ and applying Theorem 8, the required generator property follows, considering one works over $\mathbb{G}$. ◀

3.2 Proof of Claim 1

Let $F'^{k+1,r} = F^{k+1,r} - E^{k+1}$. Note that since f is multilinear, g does not depend on X. Hence $g(F^{k+1,r}) \mid_{v_{k+1}=\Lambda} = g(F'^{k+1,r})$. We will show that $g(F'^{k+1,r}) \not\equiv 0$. We have that

$$F'^{k+1,r} = F^{k,r}(Z_{\ell(k,r)}, U_k, V_k) \# F^{k,r}\left((Z_{\ell(k,r)}, U_k, V_k) + T^{k,r}(Y_{2r})\right).$$

Again we will use $y_1, y_2, \dots, y_{2r}$ as alias for the variables $z_{\ell(k,r)+1}, z_{\ell(k,r)+2}, \dots, z_{\ell(k,r)+2r}$, respectively. Corollary 7 gives us that we can write $g = \sum_{i\in[r']} p_i q_i$, for some $r' \leq r$, where

1. $\{p_1, p_2, \dots, p_{r'}\} \subseteq \mathbb{F}[x_1, x_2, \dots, x_{2^k-1}]$ and $\{q_1, q_2, \dots, q_{r'}\} \subseteq \mathbb{F}[x_{2^k+1}, x_{2^k+2}, \dots, x_{2^{k+1}}]$ are both independent sets of polynomials.
2. $\forall a \in \mathbb{F}^{r'}$, $\sum_{i\in[r']} a_i p_i$ can be computed by an oblivious ABP of read r with variable order $x_1, x_2, .., x_{2^k-1}$.
3. $\forall a \in \mathbb{F}^{r'}$, $\sum_{i\in[r']} a_i q_i$ can be computed by an oblivious ABP of read r with variable order $x_{2^k+1}, x_{2^k+2}, \dots, x_{2^{k+1}}$.

For any $a \in \mathbb{F}^{r'}$ with $a \neq \overline{0}$, $\sum_{i\in[r']} a_i p_i \not\equiv 0$, and this sum can be computed by an oblivious ABP of read r with variable order $x_1, x_2, .., x_{2^k-1}$. Hence by induction hypothesis $\sum_{i\in[r']} a_i p_i(F^{k,r}) \not\equiv 0$. Let $\hat{p}_i = p_i(F^{k,r}(z_1, \dots, z_{\ell(k,r)}, U_k, V_k))$. The above shows that $P := \{\hat{p}_1, \hat{p}_2, \dots, \hat{p}_{r'}\}$ is an independent set of polynomials. Let $\hat{q}_i = q_i(F^{k,r}(z_1, \dots, z_{\ell(k,r)}, U_k, V_k))$. Similarly we have that $Q := \{\hat{q}_1, \hat{q}_2, \dots, \hat{q}_{r'}\}$ is an independent set of polynomials.

Since $\hat{p}_1 + \hat{p}_2 + \ldots + \hat{p}_{r'} \not\equiv 0$, there exists input $c \in \mathbb{F}^{\ell(k,r)+2k}$ so that if we let $a_i = \hat{p}_i(c)$, then $a = (a_1, a_2, \dots, a_{r'}) \neq \overline{0}$. Let $V \subseteq \mathbb{F}^{\ell(k,r)+2k}$ be the algebraic set defined by the system of equations

$$\{\hat{p}_i(z_1, \dots, z_{\ell(k,r)}, U_k, V_k) = a_i : \forall i \in [r']\}$$

We know this system has a solution namely c. Since $\mathbb{F}$ is assumed to be algebraically closed, by Exercise 1.9 p. 8 in [6], we know that each irreducible component of V has dimension at least $\ell(k,r) + 2k - r'$. Since the system is solvable there must exist at least one irreducible component, and since $r \geq 1$, $\ell(k,r) + 2k - r' \geq 3$.

Let $W \subseteq \mathbb{F}^{\ell(k,r)+2k}$ be the algebraic set defined by the equation $\sum_{i\in[r']} a_i \hat{q}_i(z_1, \dots, z_{\ell(k,r)}, U_k, V_k) = 0$. Since Q is an independent set of polynomials the l.h.s. of the above equation is a nonzero polynomial. In case the l.h.s. is a non-zero constant, then we are done. Namely, letting $b \in \mathbb{F}^{\ell(k+1,r)+2(k+1)}$ be the assignment where we

set $(z_1, \ldots, z_{\ell(k,r)}, U_k, V_k)$ to c, $y_1, \ldots, y_r$ to 0, and the remaining variables arbitrarily, would give $g(F'^{k+1,r})(b) = \sum_{i\in[r']} a_i \hat{q}_i(z_1, \ldots, z_{\ell(k,r)}, U_k, V_k)(b) \neq 0$. Otherwise, we know by Proposition 1.13 in [6], that W is a finite union of hypersurfaces each of dimension $\ell(k,r) + 2k - 1$ (these correspond to the irreducible factors of $\sum_{i\in[r']} a_i \hat{q}_i(z_1, \ldots, z_{\ell(k,r)}, U_k, V_k)$). We want to argue that $V + \operatorname{Im} T$ cannot be contained in W. Namely, to see the consequence, suppose we have $c' = c'' + T(d)$, for $c'' \in V$ and $d \in \mathbb{F}^{2r}$, with $c' \notin W$. Then letting $b \in \mathbb{F}^{\ell(k+1,r)+2k}$ be the assignment where we set $(z_1, \ldots, z_{\ell(k,r)}, U_k, V_k)$ to c'' and $Y_{2r} := d$ gives that $g(F'^{k+1,r})(b) = \sum_{i\in[r']} p_i(F^{k,r}(c''))q_i(F^{k,r}(c'' + T(d))) = \sum_{i\in[r']} \hat{p}_i(c'')\hat{q}_i(c'' + T(d)) = \sum_{i\in[r']} a_i \hat{q}_i(c') \neq 0$.

We complete the proof by showing that the Zariski-closure of $V + \operatorname{Im} T$ has dimension greater than $\dim W$.

▶ Claim 2. $\dim \overline{V + \operatorname{Im} T} = \ell(k,r) + 2k$.

Proof. As remarked upon before, for any $r'' \leq r$, $\operatorname{Im} T$ contains any r''-dimensional coordinate subspace of $\mathbb{F}^{\ell(k,r)+2k}$. Namely, by setting $y_{r+i} = \alpha_{j_i}$, for all $i \in [r]$, where $\alpha_{j_1}, \alpha_{j_2}, \ldots, \alpha_{j_r}$ are distinct elements of $S_{\ell(k,r)+2k}$, we obtain $\sum_{i\in[r]} y_i \cdot H^{k,r}(y_{r+i}) = \sum_{i\in[r]} y_i \cdot H^{k,r}(\alpha_{j_i}) = \sum_{i\in[r]} y_i \cdot e_{j_i}$, where $e_1, e_2, \ldots, e_{\ell(k,r)+2k}$ are standard basis vectors of $\mathbb{F}^{\ell(k,r)+2k}$. Hence the claim follows from Lemma 4. ◀

The above claim implies that $V + \operatorname{Im} T \not\subset W$. By the above remarks, this gives that $g(F'^{k+1,r})(b) \neq 0$, for some b. This proves Claim 1. ◀

4 A Black-Box PIT Algorithm for π-OABPs

Algorithm 1 PIT Algorithm for read r π-OABPs.

Input: Black-box access to $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ computed by a π-OABP with read r.
Output: returns **true** iff $f \equiv 0$.

1: let k be such that $2^{k-1} < n \leq 2^k$.
2: let $D = \prod_{j\in[k-1]}(\ell(j,r) + 2j)(\ell(j,r) + 2j - 1)$.
3: let S_{D+1} be an arbitrary subset of $\mathbb{F}$ (or an extension field of $\mathbb{F}$ if $|\mathbb{F}| < D + 1$) of size $D + 1$.
4: let $R = S_{D+1}^{\ell(k,r)+2k}$.
5: compute $A = F^{k,r}(R)$.
6: permute the vectors in A according to π.
7: For every $a \in A$, check whether $f(a) = 0$.
8: **return** **true** if in the previous stage no nonzero was found, **false** otherwise.

▶ **Theorem 10.** *Let $\mathbb{F}$ be an arbitrary field. Using black-box Algorithm 1 we can check deterministically in time $2^{O(r \log r \cdot \log^2 n \log\log n)}$ whether a given polynomial $f \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ computed by a read r π-OABP is identically zero or not. If $char(\mathbb{F}) > 0$, the algorithm is granted black-box access to extension fields of $\mathbb{F}$.*

Proof. By Lemma 3, we can assume wlog. that f is computed by a read r π-oblivious ABP. By Theorem 8, we see that $f \equiv 0 \Leftrightarrow f(F^{k,r}) \equiv 0$. By Proposition 1, the individual degree of variables of $f(F^{k,r})$ can be bounded by $D = \prod_{j\in[k-1]}(\ell(j,r) + 2j)(\ell(j,r) + 2j - 1)$. Correctness now follows from Lemma 5. Bounding D by $(2rk + 2k)^{2k}$, and knowing that the number of variables of $f(G^{k,r})$ is $2rk + 2k + 1$, the theorem follows by straightforward arithmetic.

We remark that the hitting set A, will be constructed over an extension field of $\mathbb{F}$ if $|\mathbb{F}| < \max(\ell(k,r)+2k, 2^k)$ or $|\mathbb{F}| < D+1$. In the former case, it is because of having enough interpolation points to define the generator. In the latter case it is in order to apply Lemma 5, as was done in the above. To work over the extension field the algorithm by Shoup [18] can be used to obtain an irreducible polynomial of degree d over $\mathbb{F}$ in time $poly(d)$. For us, it suffices for the degree of this polynomial to be bounded by $O(\log n \log r + \log n \log\log n)$. Field operations in the extension field then take time $poly(\log n, \log r)$, assuming a unit cost model for operations in $\mathbb{F}$. The cost of constructing A this way, can easily be seen to be subsumed by the time bound given in the theorem. ◀

The above implies that read $polylog(n)$ π-OABPs can be tested in DTIME$[2^{O(polylog(n))}]$.

5 Separation Results and Lower Bounds for OABPs

Omitted proofs in this section will appear in the full version of the paper.

▶ **Theorem 11.** *Any OABP computing the permanent or determinant of an $n \times n$ matrix of variables has size $\Omega(2^n/n)$ and read $\Omega(2^n/n^2)$.*

By extending the construction in [15], we can prove the following theorem:

▶ **Theorem 12.** *Let $X = \{x_i\}_{i\in[2n+1]}$ and $\mathcal{W} = \{w_{i,j,k}\}_{i,j,k\in[2n+1]}$ be sets of variables. We can construct an explicit polynomial $p \in \mathbb{F}[X, \mathcal{W}]$ such that any OABP A over variables $X \cup \mathcal{W}$ using constants from $\mathbb{F}$ computing p requires some variable to be read at least 2^n times.*

We can also reinterpret the above result to be giving a stronger lower bound (seen as a function of the number of variables), but for a polynomial which uses $O(n^3)$ transcendental constants in its definition.

▶ **Corollary 13.** *For any field $\mathbb{F}$, and any extension field $\mathbb{G}$ of $\mathbb{F}$ of transcendence degree at least $(2n+1)^3$, there exists an explicit polynomial $p \in \mathbb{G}[x_1, x_2, \ldots, x_{2n+1}]$, such that any OABP over $\mathbb{G}$ computing p requires some variable to be read at least 2^n times.*

Consider the elementary symmetric polynomial $S_n^k = \sum_{S\subset[n],|S|=k} \prod_{i\in S} x_i$.

▶ **Theorem 14.** *S_n^k can not be computed by an R_{k-1}-OABP, for $n \geq 2k-1$, $k \geq 2$.*

▶ **Theorem 15.** *S_n^k can be computed by an R_k-OABP of size $O(kn)$, for $n \geq k \geq 1$.*

The following theorem shows that under different permutations π and π', the gap between the number of reads for the models π-OABP and π'-OABP can be exponentially large.

▶ **Theorem 16.** *Given $X = \{x_0, x_1, \ldots, x_{2n-1}, x_{2n}\}$, $n \geq 1$, there exists a polynomial p on X, and two permutations π and π' on X, such that 1) There exists a read-once π-OABP computing p, and 2) Any π'-OABP computing p requires read 2^n.*

References

1 M. Agrawal. Proving lower bounds via pseudo-random generators. In *Proc. 25th Annual Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 92–105, 2005.

2 M. Agrawal and V. Vinay. Arithmetic circuits: A chasm at depth four. In *Proc. 49th FOCS*, pages 67–75, 2008.

3 N. Alon. Combinatorial nullstellensatz. *Combinatorics, Probability and Computing*, 8(1–2):7–29, 1999.

4 R.E. Bryant. On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Trans. Computers*, 40(2):205–213, 1991.

5 D. Cox, J. Little, and D. O'Shea. *Ideals, Varieties, and Algorithms, Second Edition*. Undergraduate Texts in Mathematics. Springer Verlag, 1996.

6 R. Hartshorne. *Algebraic Geometry*. Graduate Texts in Mathematics, Vol 52. Springer Verlag, 1977.

7 R. Impagliazzo, N. Nisan, and A. Wigderson. Pseudorandomness for network algorithms. In *Proc. 26th STOC*, pages 356–364, 1994.

8 M. Jansen. Lower bounds for syntactically multilinear algebraic branching programs. In *Proc. 33rd MFCS*, volume 5162 of *Lect. Notes in Comp. Sci.*, pages 407–418, 2008.

9 M. Jansen. Weakening assumptions for deterministic subexponential time non-singular matrix completion. In *27th STACS*, volume 5 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 465–476, 2010.

10 M. Jansen, Y. Qiao, and J. Sarma M.N. Deterministic identity testing of read-once algebraic branching programs, 2009. http://arxiv.org/abs/0912.2565.

11 V. Kabanets and R. Impagliazzo. Derandomizing polynomial identity testing means proving circuit lower bounds. *Computational Complexity*, 13(1–2):1–44, 2004.

12 N. Nisan. Lower bounds for non-commutative computation: extended abstract. In *Proc. 23rd Annual ACM STOC*, pages 410–418, 1991.

13 R. Raz. Multilinear formulas for permanent and determinant are of super-polynomial size. *J. Assn. Comp. Mach.*, 56(2):1–17, 2009.

14 R. Raz and A. Shpilka. Deterministic polynomial identity testing in non commutative models. *Computational Complexity*, 14(1):1–19, 2005.

15 R. Raz and A. Yehudayoff. Balancing syntactically multilinear arithmetical circuits. *Computational Complexity*, 17(4):515–535, 2008.

16 N. Saxena. Progress of polynomial identity testing. Technical Report ECCC TR09-101, Electronic Colloquium in Computational Complexity, 2009.

17 J.T. Schwartz. Fast probabilistic algorithms for polynomial identities. *J. Assn. Comp. Mach.*, 27:701–717, 1980.

18 V. Shoup. New algorithms for finding irreducible polynomials over finite fields. In *Proc. 29th FOCS*, pages 283–290, 1988.

19 A. Shpilka and I. Volkovich. Improved polynomial identity testing of read-once formulas. In *Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques, volume 5687 of LNCS*, pages 700–713, 2009.

20 S. Toda. Classes of arithmetic circuits capturing the complexity of computing the determinant. *IEICE Trans. Inf. Syst.*, E75-D:116–124, 1992.

21 L. Valiant. Completeness classes in algebra. Technical Report CSR-40-79, Dept. of Computer Science, University of Edinburgh, April 1979.

22 R. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposium on Symbolic and Algebraic Manipulation (EUROSAM '79)*, volume 72 of *Lect. Notes in Comp. Sci.*, pages 216–226. Springer Verlag, 1979.

Computing Rational Radical Sums in Uniform TC^0

Paul Hunter[1], Patricia Bouyer[2], Nicolas Markey[2], Joël Ouaknine[1], and James Worrell[1]

1 **Oxford University Computing Laboratory, UK**
{paul.hunter,joel.ouaknine,james.worrell}@comlab.ox.ac.uk
2 **Lab. Spécification et Vérification, CNRS & ENS Cachan, France**
{bouyer,markey}@lsv.ens-cachan.fr

Abstract

A fundamental problem in numerical computation and computational geometry is to determine the sign of arithmetic expressions in radicals. Here we consider the simpler problem of deciding whether $\sum_{i=1}^{m} C_i A_i^{X_i}$ is zero for given rational numbers A_i, C_i, X_i. It has been known for almost twenty years that this can be decided in polynomial time [2]. In this paper we improve this result by showing membership in uniform TC^0. This requires several significant departures from Blömer's polynomial-time algorithm as the latter crucially relies on primitives, such as gcd computation and binary search, that are not known to be in TC^0.

Keywords and phrases Sum of square roots, Threshold circuits, Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.308

1 Introduction

The SqrtSum problem is as follows: given integers $a_1, \ldots, a_n, b_1, \ldots, b_m$, is it the case that $\sum_{i=1}^{n} \sqrt{a_i} - \sum_{i=1}^{m} \sqrt{b_i} > 0$? This problem naturally arises in computational geometry whenever one needs to compare the length of two paths, as in the *Euclidean Travelling Salesman Problem* [8, 14] for example.

SqrtSum can be solved in polynomial time on a unit-cost RAM, i.e., counting only the number of algebraic operations [18]: one simply uses numerical methods to compute each root to a sufficient degree of accuracy. However the problem is not known to be in P when one accounts for the bit complexity of arithmetic operations. Hence certain ostensibly simple problems in computational geometry, such as computing minimal spanning trees, are not known to be in P. Moreover since SqrtSum is not even known to be in NP it is also not known whether the Euclidean Travelling Salesman problem is in NP.

The difficulty in solving SqrtSum hinges on the fact that the best root separation bounds to hand require that one compute a super-polynomial number of bits of the expression $\sum_{i=1}^{n} \sqrt{a_i} - \sum_{i=1}^{m} \sqrt{b_i}$ to determine its sign. The question of determining optimal separation bounds was posed at least as far back as [13]. More recent work on the problem includes [15, 16, 3, 5]; also [12] presents a conjecture that would imply P-membership of SqrtSum.

SqrtSum has found applications in numerical decision problems outside the area of computational geometry. For instance, it has recently been used as a complexity lower bound for several problems related to recursive probabilistic systems. Etessami and Yannakakis [7] show that SqrtSum is reducible in polynomial time to the problem of determining whether a stochastic context-free grammar produces a terminal string with probability greater than a given threshold. This latter problem is in turn equivalent to the reachability problem for a certain subclass of probabilistic pushdown automata [7]. In another paper Etessami and Yannakakis [6] consider a range of algorithmic problems in game theory and economics,

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 308–316

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

several of which are shown to be as hard as SqrtSum. For example, SqrtSum can be reduced to various decision problems concerning Nash Equilibria and the value of Shapley stochastic games.

Since the decision problem for the existential fragment of the first-order theory of real-closed fields is known to be in Pspace [4], it is straightforward that SqrtSum is also in Pspace. A more general problem than SqrtSum is the problem PosSLP of determining whether an arithmetic circuit over the basis $\{+, -, *\}$ evaluates to a positive integer. PosSLP was shown to lie in the 4th level of the counting hierarchy [1]. To the best of our knowledge this result also yields the best upper bound for SqrtSum.

The subject of this paper involves upper bounds for the easier problem SqrtSumEQ: given integers $a_1, \ldots, a_n, b_1, \ldots b_m$, does $\sum_{i=1}^{n} \sqrt{a_i} - \sum_{i=1}^{m} \sqrt{b_i} = 0$? This problem is also of key importance in numerical analysis and exact geometric computation, as discussed in [20]. SqrtSumEQ is apparently more tractable than SqrtSum; a polynomial-time decision procedure has been given by Blömer [2]. In fact [2] gives a polynomial-time algorithm for a more general problem in which one considers arbitrary integer roots rather than just square roots, and with arbitrary rational coefficients in front of the radicals (i.e., not just 1 or -1 as in SqrtSumEQ).

In this paper we consider a further generalisation of SqrtSumEQ, where we allow any rational exponent (less than 1) rather than exponents of the form $\frac{1}{N}$ for N a positive integer. In particular, we are interested in the following problem:

RadicalSumEQ	
Instance:	Rational numbers C_i, A_i, X_i for $1 \leq i \leq m$ with $A_i > 0$ and $0 \leq X_i \leq 1$
Problem:	Does $\sum_{i=1}^{m} C_i A_i^{X_i} = 0$?

Our main result is that RadicalSumEQ has very low complexity within P, and can be solved with fixed-depth circuits consisting of AND, OR, and threshold gates of unbounded fan-in:

▶ **Theorem 1.** RadicalSumEQ $\in$ uniform TC^0.

The notion of uniformity referred to above is DLogtime-uniformity, which is the strongest uniformity requirement that is generally applicable.

Our TC^0 procedure adapts Blömer's polynomial-time algorithm for comparing radical expressions [2] and exploits the fact that division and iterated multiplication are in uniform TC^0; see [9, 10]. However we depart from [2] in several critical respects. Firstly [2] only considers the case of exponents of the form $\frac{1}{N}$. Whilst $A^{\frac{M}{N}}$ can be rewritten as $(A^M)^{\frac{1}{N}}$, the rational A^M cannot in general be explicitly computed in polynomial time and it is not clear how to apply Blömer's algorithm without doing so. Secondly at various points Blömer's algorithm requires the computation of the greatest common divisor of two numbers (specifically, denominators in the exponents) and binary search (to find integer d-th roots), two techniques not known to be in TC^0: indeed it is an open problem whether gcd computation is even in NC^1 [11].

One of the consequences of our work is that, unless $\mathrm{TC}^0 = \mathrm{P}$, SqrtSumEQ has strictly lower complexity than EqSLP, the problem of determining whether an arithmetic circuit over the basis $\{+, -, *\}$ evaluates to zero or not. Indeed, the latter is easily seen to be P-hard, by reduction from the circuit value problem. In contrast, it is still open whether SqrtSum has the same complexity as PosSLP or not.

The paper is organised as follows. In Section 2 we recall the definitions and notation we use throughout the paper. In Section 3 we present two procedures necessary for our main

algorithm. The first of these procedures is a restatement of a known result but presented in a form suitable for our needs. The second shows how to compute the ratio of two radicals in uniform TC^0, a result which we believe to be of independent interest. In Section 4 we present our main algorithm for deciding RadicalSumEQ, and in Section 5 we discuss further extensions of the problem.

2 Preliminaries

Throughout this paper we assume familiarity with standard notions of circuit complexity; an excellent reference on the subject is [19].

Recall that a circuit family $\{\mathsf{C}_n\}$ is DLogtime-uniform if there is a deterministic Turing machine that, given n and the name of a gate g, can determine g's label and neighbours in time $O(\log n)$.

In the sequel we always use n to represent the input size. In particular, we assume integers provided as part of the input to have absolute value bounded above by 2^n, and for there to be at most n terms in the sum (that is, $m \leq n$). This means the actual input size is $O(n^2)$, however this does not affect the overall result as the class TC^0 is closed under polynomial changes in input size. As is customary in this area of circuit complexity, we call $n^{O(1)}$-bit numbers *large* and denote them with uppercase letters, and we say $(\log n)^{O(1)}$-bit numbers are *small* and use lowercase letters to represent them.

We assume rational numbers are represented as ratios of two (not necessarily co-prime) integers. Whilst we do not require the rational numbers to be in reduced form, we use the fact that the size required to represent a rational number is bounded below by the size of its reduced form. For $A \in \mathbb{Q}$, we define $\|A\| := |M \cdot N|$ where $\frac{M}{N} = A$ and $\gcd(M, N) = 1$. The *height* of A is defined as $\mathrm{ht}(A) := 1 + \log \|A\|$. It is clear that $\|A\| = \min |M \cdot N|$ where the minimum is taken over all representations of A as $\frac{M}{N}$, thus the height of A provides a lower bound on the number of bits required to represent A.

The following properties of $\|\cdot\|$ will prove useful:

▶ **Lemma 2.** *For $A, B \in \mathbb{Q}$ and $X, Y \in \mathbb{R}$:*

1. *If $A^X \in \mathbb{Q}$ then $\left\|A^X\right\| = \|A\|^{|X|}$, in particular $\left\|\frac{1}{A}\right\| = \|A\|$.*
2. *If $A^X \cdot B^Y \in \mathbb{Q}$ then $\left\|A^X \cdot B^Y\right\| \leq \|A\|^{|X|} \cdot \|B\|^{|Y|}$.*

Proof. For the first result, consider first $X \geq 0$. Let $A = \frac{M}{N}$ where $\gcd(M, N) = 1$. Clearly, $\left\|A^X\right\| = |M^X \cdot N^X| = |M \cdot N|^X = \|A\|^X$. To extend the result to $X < 0$ we observe from the symmetry of the definition of $\|\cdot\|$ that $\|A\| = \left\|A^{-1}\right\|$. Hence $\left\|A^X\right\| = \left\|(A^{-1})^{|X|}\right\| = \left\|A^{-1}\right\|^{|X|} = \|A\|^{|X|}$.

For the second result, we observe that for $A = 0$ the result holds trivially and for $A = 1$ the result follows from the first part of the proof. So assume $A \neq 0, 1$ and let $c = \frac{\log B}{\log A}$. Since $A^c = B$, it follows from the above result that $\|A\|^{|c|} = \|A^c\| = \|B\|$. Therefore,

$$\begin{aligned}
\left\|A^X \cdot B^Y\right\| &= \left\|A^{X+cY}\right\| \\
&= \|A\|^{|X+cY|} && \text{(from the first result)} \\
&\leq \|A\|^{|X|+|c|\cdot|Y|} && \text{(by the triangle inequality)} \\
&= \|A\|^{|X|} \cdot \|B\|^{|Y|} && \text{as required.}
\end{aligned}$$

◀

We will make use of some standard parallel algorithms and techniques known to be computable in uniform TC^0, notably:

- Existential guessing between $n^{O(1)}$ choices (in particular, guessing small integers),
- Universal (parallel) computation amongst $n^{O(1)}$ choices,
- Addition (and subtraction) of n n-bit numbers, and
- Iterated multiplication of n n-bit numbers and integer division of two n-bit numbers [9].

For ease of reference, in each *term* $C_i A_i^{X_i}$, C_i is the *coefficient*, A_i is the *base*, X_i is the *exponent* and $A_i^{X_i}$ is the *radical* (even though it may be rational).

3 TC^0 Tools

In this section we present two TC^0-procedures necessary for our algorithm. The first of these concerns determining whether an integer root of a given rational is itself rational, and if so, computing the root. The result follows from observations in [10] and [17] that functions given by a convergent power series, in particular $A^{\frac{1}{k}}$, can be approximated using iterated multiplication. We present it here in a form appropriate for our needs: computing the value if it is rational or failing if it is not rational. The algorithm uses the power series approximation to compute an approximant to sufficiently high accuracy and then tests if this estimation is the true value of the root by exponentiation. As we are dealing with rationals, it initially appears that some care is needed in extracting the approximant. However, as $\mathrm{ht}(A^{\frac{1}{k}}) \leq \mathrm{ht}(A)$, if $A^{\frac{1}{k}}$ is rational, its binary expansion will repeat after $O(\mathrm{ht}(A))$ bits. Thus to find an approximant it suffices to compute polynomially many bits to find the period of its binary expansion and then compute the rational using standard techniques. The situation appears clearer in the case of the integers: after sufficiently many computed bits we truncate our approximation and consider the integers around the value computed; but this is simply the rational case shifted by a factor of $2^{O(n)}$. Nevertheless, as the technique for extracting the approximant in the integer case is simpler, we adopt this procedure (see Lemma 3) for root computation and extend it to rationals (in Algorithm 2) by rationalising the denominator.

▶ **Lemma 3.** *Let a be a $(\log n)$-bit positive integer and B be an n-bit positive integer. There exists a uniform TC^0-algorithm which computes $\sqrt[a]{B}$ if it is an integer or fails if it is not an integer.*

Proof. As mentioned above, the idea of the algorithm (presented in Algorithm 1) is to compute an integer approximation (technically three approximations) to $\sqrt[a]{B}$ and then check if the a-th power of the approximant is equal to B. The steps which are not clearly in uniform TC^0 are the computation of the first n bits of $\sqrt[a]{B}$, and the evaluation of the antecedent in the **if** statement. Membership of TC^0 for the computation of R follows from the result of [10] (Corollary 6.5) that polynomially many bits of $X^{\frac{1}{k}}$ can be computed in uniform TC^0. For the antecedent, iterated multiplication can be used to compute $\hat{R}^a$, $(\hat{R}+1)^a$, and $(\hat{R}-1)^a$ in uniform TC^0. To show correctness, we observe that as $a \geq 1$, $\mathrm{ht}(\sqrt[a]{B}) \leq \mathrm{ht}(B)$, and so $\sqrt[a]{B}$ requires at most n bits if it is an integer. Thus $|\hat{R} - \sqrt[a]{B}| \leq 1$, and the only possible integral values for $\sqrt[a]{B}$ are $\hat{R}-1$, $\hat{R}$, or $\hat{R}+1$. ◀

Although our final algorithm does not require the computation of large roots, the extension is trivial as a consequence of the following observation.

▶ **Lemma 4.** *Let $A \in \mathbb{Z}$, $B \in \mathbb{Q}$, $A, B > 0$, $B \neq 1$. If $\sqrt[A]{B} \in \mathbb{Q}$ then $A < \mathrm{ht}(B)$.*

Proof. Let $B = \frac{M}{N}$ where $\gcd(M,N) = 1$. As $B \neq 1$ there exists a prime p such that $p|M$ or $p|N$. Assume without loss of generality $p|M$. As $\sqrt[A]{B} = \frac{\sqrt[A]{M}}{\sqrt[A]{N}}$ is rational we have $p^A|M$. As $M < 2^{\mathrm{ht}(B)}$ and $p \geq 2$, it follows that $A < \mathrm{ht}(B)$. ◀

Algorithm 1 Computing integer roots

Input: $n, a, B \in \mathbb{Z}$, $0 < a < n$, $0 \leq B < 2^n$
Returns: $\sqrt[a]{B}$ or Fail if $\sqrt[a]{B} \notin \mathbb{Z}$.
  Compute R, the first n bits of $\sqrt[a]{B}$
  Let $\hat{R} = \lfloor R \rfloor$
  if $(\hat{R}-1)^a = B$ **or** $\hat{R}^a = B$ **or** $(\hat{R}+1)^a = B$ **then**
    return $(\hat{R}-1)$, $\hat{R}$, or $(\hat{R}+1)$ as appropriate
  else
    return Fail
  end if

Our algorithm for computing roots of rationals is presented in Algorithm 2.

▶ **Proposition 5.** *For $A \in \mathbb{Z}$, $B \in \mathbb{Q}$, $A, B > 0$ there exists a uniform TC^0-algorithm which computes $\sqrt[A]{B}$ if it is rational or fails if it is irrational.*

Algorithm 2 Computing rational roots

Input: $A \in \mathbb{Z}$, $B = \frac{M}{N} \in \mathbb{Q}$, $A, B > 0$
Returns: $\sqrt[A]{B}$ or Fail if $\sqrt[A]{B} \notin \mathbb{Q}$.
  if $B = 1$ **then**
    return 1
  else if $A \geq \mathrm{ht}(B)$ **then**
    return Fail
  else
    Compute $C = \sqrt[A]{M \cdot N^{A-1}}$
    if $C \notin \mathbb{Z}$ **then**
      return Fail
    end if
    return $\frac{C}{N}$
  end if

The second algorithm of this section, presented in Algorithm 3, overcomes the difficulties with Blömer's procedure for computing the ratio of two radicals.

The core of the correctness of Algorithm 3 is presented in the following lemma; if the ratio of two radicals is rational then one of two cases occurs, either the bases are powers of some common base, or the exponents have low height relative to their value. It is this case split that forms the basis of our algorithm: in the first case we can existentially guess the powers of the common base, and in the second we can guess reduced forms for X and Y and apply the algorithm of Blömer.

▶ **Lemma 6.** *For $A, B, X, Y \in \mathbb{Q}_{>0}$ if $\frac{A^X}{B^Y} \in \mathbb{Q}$ then either:*
- *There exists $Q \in \mathbb{Q}$ and $\alpha, \beta \in \mathbb{Z}$ with $\alpha X - \beta Y \in \mathbb{Z}$ such that $A = Q^\alpha$ and $B = Q^\beta$, or*
- $\|X\| < 4\mathrm{ht}(A)\mathrm{ht}(B)^2(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B))$ *and*
  $\|Y\| < 4\mathrm{ht}(A)^2\mathrm{ht}(B)(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B))$.

Proof. Suppose $\frac{A^X}{B^Y} = M \in \mathbb{Q}$. From Lemma 2 we observe that $\|M\| = \left\|\frac{A^X}{B^Y}\right\| \leq \|A\|^X \|B\|^Y$, so $\mathrm{ht}(M) < X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B)$. Let $A = \prod p_i^{a_i}$, $B = \prod p_i^{b_i}$ and $M = \prod p_i^{m_i}$ where for all i, p_i is prime and $a_i, b_i, m_i \in \mathbb{Z}$. We observe that $|a_i| < \mathrm{ht}(A)$, $|b_i| < \mathrm{ht}(B)$ and $|m_i| < \mathrm{ht}(M)$.

Algorithm 3 Computing rational radical ratios

Input: $A, B, X, Y \in \mathbb{Q}$, $A, B > 0$, $X, Y \in [0, 1]$
Returns: $\frac{A^X}{B^Y}$ or Fail if $\frac{A^X}{B^Y} \notin \mathbb{Q}$

```
Let n = max{ht(A), ht(B), ht(X), ht(Y)}
Existentially guess non-negative integers a, b < n
if aX − bY ∈ ℤ and Q = ᵃ√A = ᵇ√B ∈ ℚ† then
    return Q^{aX−bY}
end if
Existentially guess non-negative integers x, x′, y, y′ < 8n⁴
if X = x/x′ and Y = y/y′ then
    Let z = gcd(x′, y′), x″ = x′/z, and y″ = y′/z
    Let R_A = ˣ″√A^x and R_B = ʸ″√B^y
    if R_A ∈ ℚ and R_B ∈ ℚ and R = ᶻ√(R_A/R_B) ∈ ℚ then
        return R
    end if
end if
return Fail
```

† We allow the equality to hold here if $a = 0$ and $A = 1$ or if $b = 0$ and $B = 1$, setting $Q = 1$ if $a = b = 0$ and $A = B = 1$.

Consider the (integral) vectors $\mathbf{a} = (a_i)$, $\mathbf{b} = (b_i)$, and $\mathbf{m} = (m_i)$. By equating prime powers we have

$$\mathbf{a}X = \mathbf{m} + \mathbf{b}Y \cdot \qquad (*)$$

We consider two cases.

Case 1: $\mathbf{a}$ and $\mathbf{b}$ are linearly dependent (over $\mathbb{Q}$). In this case, there exist integers k and l (not necessarily co-prime) and an integral vector $\mathbf{q} = (q_i)$ such that $a_i = k \cdot q_i$, $b_i = l \cdot q_i$ and the q_i have no common factor. From $(*)$, $\mathbf{m} = (kX - lY)\mathbf{q}$. As $\mathbf{m}$ is integral and the q_i have no common factor, it follows that $(kX - lY) = c \in \mathbb{Z}$. Setting $Q = \prod p_i^{q_i}$ we have $A = Q^k$, $B = Q^l$, $M = Q^c$ and $kX = c + lY$.

Case 2: $\mathbf{a}$ and $\mathbf{b}$ are linearly independent (over $\mathbb{Q}$). In this case, there exist $i \neq j$ such that the vectors (a_i, a_j) and (b_i, b_j) are linearly independent. It therefore follows that X and Y satisfying $(*)$ are unique. Indeed $X = \frac{b_i m_j - b_j m_i}{b_i a_j - b_j a_i}$ and $Y = \frac{a_i m_j - a_j m_i}{b_i a_j - b_j a_i}$. Thus

$$\begin{aligned}
\|X\| &= \left\| \frac{b_i m_j - b_j m_i}{b_i a_j - b_j a_i} \right\| \\
&\leq \|b_i m_j - b_j m_i\| \cdot \|b_i a_j - b_j a_i\| \\
&= |b_i m_j - b_j m_i| \cdot |b_i a_j - b_j a_i| \\
&\leq (|b_i m_j| + |b_j m_i|)(|b_i a_j| + |b_j a_i|) \quad \text{(by the triangle inequality)} \\
&< (2\mathrm{ht}(B)\mathrm{ht}(M))(2\mathrm{ht}(B)\mathrm{ht}(A)) \\
&< 4\mathrm{ht}(A)\mathrm{ht}(B)^2(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B)),
\end{aligned}$$

and likewise $\|Y\| < 4\mathrm{ht}(A)^2\mathrm{ht}(B)(X{\cdot}\mathrm{ht}(A) + Y{\cdot}\mathrm{ht}(B))$. ◀

When Case 1 of Lemma 6 holds, it is clear Algorithm 3 correctly computes the ratio $\frac{A^X}{B^Y}$; the bounds on X and Y ensure that Q^{aX-bY} can be evaluated with iterated multiplication. To complete the correctness result we need to show the correctness of the algorithm when

Case 2 of the above lemma occurs. This follows directly from the following result observed by Blömer [2]:

▶ **Lemma 7.** *For $q_1, q_2 \in \mathbb{Q}$, $d_1, d_2 \in \mathbb{N}$, the following are equivalent:*

- $\frac{\sqrt[d_1]{q_1}}{\sqrt[d_2]{q_2}} \in \mathbb{Q}$
- *If $d = \gcd(d_1, d_2)$ then*
  - $r_i = \sqrt[d_i]{q_i^d} \in \mathbb{Q}$ *for $i = 1, 2$, and*
  - $\sqrt[d]{\frac{r_1}{r_2}} \in \mathbb{Q}$.

It is straightforward to show that Algorithm 3 can be implemented with a uniform TC^0 circuit. The non-obvious steps are in the computation of Q, R_A, R_B and R where we use Algorithm 2, and the computation of z which can be calculated because x' and y' are small[1]. Lemmas 6 and 7 establish the correctness of the algorithm, giving us the following:

▶ **Theorem 8.** *Let $A, B, X, Y \in \mathbb{Q}$, with $A, B > 0$, $X, Y \in [0, 1]$. There exists a uniform* TC^0*-algorithm which computes the ratio $\frac{A^X}{B^Y}$ if it is rational, or fails if it is irrational.*

We note that in Theorem 8 we only need the upper bound on X and Y to compute the ratio: in the first case of Lemma 6 the bound ensures that Q^{aX-bY} is computable using iterated multiplication, and in the second case the bound ensures that $\|X\|$ and $\|Y\|$ are small. If instead we were provided with the ratio M and simply asked to check if $A^X = MB^Y$, we no longer require the bound on X and Y: in the first case $aX - bY \leq \mathrm{ht}(M)$ and in the second case we can use the bounds obtained in terms of $\mathrm{ht}(A)$, $\mathrm{ht}(B)$, and $\mathrm{ht}(M)$. This gives us the following additional result:

▶ **Theorem 9.** *For $A, B, M, X, Y \in \mathbb{Q}_{\geq 0}$, whether $A^X = MB^Y$ can be decided in uniform* TC^0.

4 Deciding RadicalSumEQ in TC^0

In this section we present our TC^0-algorithm for deciding RadicalSumEQ. Critical to our algorithm is the following result presented in Blömer:

▶ **Lemma 10** (Theorem 4 of [2])**.** *For $\rho_i \in \mathbb{Q}$, $d_i \in \mathbb{N}$, $1 \leq i \leq m$, the radicals $\sqrt[d_1]{\rho_1}, \ldots, \sqrt[d_m]{\rho_m}$ are linearly independent over $\mathbb{Q}$ if they are pairwise linearly independent.*

Clearly this result extends to arbitrary rational exponents, giving the following procedure (also presented in [2]) for determining if $S = \sum_{i=1}^{m} C_i A_i^{X_i} = 0$. Using Algorithm 3 partition the terms of S into linearly dependent groups $S_1, \ldots, S_{m'}$. For convenience let us assume $C_i A_i^{X_i} \in S_i$ for $1 \leq i \leq m'$. Again using Algorithm 3, replace each term in each group by the rational multiple of some common radical. For example, if $C_j A_j^{X_j} \in S_i$, replace it with $C_j R_{ij} A_i^{X_i}$ where $R_{ji} = \frac{A_j^{X_j}}{A_i^{X_i}}$ is computed with Algorithm 3. Then S can be written as

$$S = \sum_{i=1}^{m'} \left(\sum_j C_j R_{ji} \right) A_i^{X_i}$$

where j in the inner sum runs over all indices of terms in S_i. From the above result, as $A_1^{X_1}, \ldots, A_{m'}^{X_{m'}}$ are pairwise linearly independent, they form a linearly independent set. Thus

[1] Existentially guess z and verify in parallel that it is the greatest of all common divisors of x' and y'.

$S = 0$ if and only if $\sum_j C_j \cdot R_{ji} = 0$ for all i, $1 \leq i \leq m'$, and this is easily checked. To simplify the parallelisation of this algorithm, rather than gathering linearly dependent terms under a common radical, we treat each radical as the common radical, repeating the coefficient check several times. The full algorithm is specified in Algorithm 4.

Algorithm 4 Deciding RadicalSumEQ

Input: $\{A_i, C_i, X_i : 1 \leq i \leq m\} \subseteq \mathbb{Q}$ with $A_i > 0$ and $X_i \in [0, 1]$
Returns: True if and only if $\sum_{i=1}^{m} C_i A_i^{X_i} = 0$

```
for all i, j ≤ m do
  Let R_ij = A_i^{X_i} / A_j^{X_j}
  if R_ij ∉ ℚ then
    Let R_ij = 0
  end if
end for
for all j ≤ m do
  if Σ_{i=1}^{m} C_i R_ij ≠ 0 then
    return False
  end if
end for
return True
```

The only step of Algorithm 4 which is not clearly in uniform TC^0 is the computation of R_{ij}. Theorem 8 establishes its membership in TC^0. The correctness of the algorithm follows from Lemma 10 and the discussion above. Combining these together gives our main result.

▶ **Theorem 1.** RadicalSumEQ $\in$ uniform TC^0.

5 Further Work

It is clear from Lemma 6 that we can extend Algorithm 3 (and hence Algorithm 4) to exponents bounded (in value) by some polynomial in n. This raises the question of whether we can remove the upper bound on the exponents completely. Theorem 9 shows that we can do so in the special case where $m = 2$. By rewriting A^X as $A^{\lfloor X \rfloor} \cdot A^{\{X\}}$ where $\{X\}$ denotes the fractional part of X, we can absorb the "rational part" $A^{\lfloor X \rfloor}$ of the radical into the coefficient, and run our algorithm up to the point where we check if $\sum_{i=1}^{m} C_i R_{ij} = 0$. Thus we have reduced the problem to deciding if a given rational-valued point is a root of a sparse, multivariate polynomial (a natural sub-instance of the unbounded version of RadicalSumEQ). Whether or not this problem is in TC^0, or even in P, is part of ongoing work.

References

1 Eric Allender, Peter Bürgisser, Johan Kjeldgaard-Pedersen, and Peter Bro Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

2 Johannes Blömer. Computing sums of radicals in polynomial time. In *Proceedings of the 32nd Annual Symposium on Foundations of Computer Science*, pages 670–677. IEEE, 1991.

3 C. Burnikel, S. Funke, K. Mehlhorn, S. Schirra, and S. Schmitt. A separation bound for real algebraic expressions. *Algorithmica*, 55(1):14–28, 2009.

4 John F. Canny. Some algebraic and geometric computations in PSPACE. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, pages 460–467. ACM, 1988.

5 Qi Cheng, Xianmeng Meng, Celi Sun, and Jiazhe Chen. Bounding the sum of square roots via lattice reduction. *Math. Comput. (to appear)*, 2010.

6 Kousha Etessami and Mihalis Yannakakis. On the complexity of Nash equilibria and other fixed points (extended abstract). In *Proceedings of the 48th Annual IEEE Symposium on Foundations of Computer Science*, pages 113–123. IEEE Computer Society, 2007.

7 Kousha Etessami and Mihalis Yannakakis. Recursive Markov chains, stochastic grammars, and monotone systems of nonlinear equations. *J. ACM*, 56(1):1–66, 2009.

8 M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

9 William Hesse. Division is in uniform TC^0. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming*, volume 2076 of *Lecture Notes in Computer Science*, pages 104–114. Springer, 2001.

10 William Hesse, Eric Allender, and David A. Mix Barrington. Uniform constant-depth threshold circuits for division and iterated multiplication. *J. Comput. Syst. Sci.*, 65:695–716, 2002.

11 Yu Lin-Kriz and Victor Y. Pan. On parallel complexity of integer linear programming, gcd and the iterated mod function. In *Proceedings of the Third Annual Symposium on Discrete Algorithms*, pages 124–137. ACM/SIAM, 1992.

12 Gregorio Malajovich. An effective version of Kronecker's theorem on simultaneous Diophantine approximation. *Preprint*, 2001.

13 Joseph O'Rourke. Advanced problem 6369. *Amer. Math. Monthly*, 88(10):769, 1981.

14 Christos H. Papadimitriou. The Euclidean traveling salesman problem is NP-complete. *Theor. Comput. Sci.*, 4(3):237–244, 1977.

15 Sylvain Pion and Chee Yap. Constructive root bound method for k-ary rational input numbers. *Theor. Comput. Sci.*, 369(1-3):361–376, 2006.

16 Jianbo Qian and Cao An Wang. How much precision is needed to compare two sums of square roots of integers? *Inf. Process. Lett.*, 100(5):194–198, 2006.

17 John H. Reif and Stephen R. Tait. On threshold circuits and polynomial computation. *SIAM J. of Comput.*, 21:896–908, 1992.

18 Prasoon Tiwari. A problem that is easier to solve on the unit-cost algebraic RAM. *J. Complexity*, 8(4):393–397, 1992.

19 Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer, 1999.

20 Chee K. Yap. Robust geometric computation. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 41, pages 927–952. Chapman & Hall/CRC, 2nd edition, 2004.

Graph Isomorphism is not AC^0 reducible to Group Isomorphism

Arkadev Chattopadhyay[1], Jacobo Torán[2], and Fabian Wagner[2]

1 **Department of Computer Science, University of Toronto, ON M5S 3G4 Canada**
arkadev@cs.toronto.edu

2 **Institut für Theoretische Informatik, Universität Ulm, 89069 Ulm, Germany**
{jacobo.toran, fabian.wagner}@uni-ulm.de

Abstract

We give a new upper bound for the Group and Quasigroup Isomorphism problems when the input structures are given explicitly by multiplication tables. We show that these problems can be computed by polynomial size nondeterministic circuits of unbounded fan-in with $O(\log \log n)$ depth and $O(\log^2 n)$ nondeterministic bits, where n is the number of group elements. This improves the existing upper bound from [Wol94] for the problems. In the previous upper bound the circuits have bounded fan-in but depth $O(\log^2 n)$ and also $O(\log^2 n)$ nondeterministic bits. We then prove that the kind of circuits from our upper bound cannot compute the Parity function. Since Parity is AC^0 reducible to Graph Isomorphism, this implies that Graph Isomorphism is strictly harder than Group or Quasigroup Isomorphism under the ordering defined by AC^0 reductions.

Keywords and phrases Complexity, Algorithms, Group Isomorphism Problem, Circuit Complexity

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.317

1 Introduction.

The input of the Group Isomorphism problem GroupIso consists of two groups G_1 and G_2 of order n given by multiplication tables ($n \times n$ matrices of integers between 1 and n) and it is asked whether the groups are isomorphic, that is, whether there is a bijection φ between the elements of both groups satisfying for every pair of elements i, j, $\varphi(ij) = \varphi(i)\varphi(j)$ (for convenience, we represent in both groups the group operation by concatenation). A quasigroup is an algebraic structure $(\Omega, \cdot)$ where the set Ω is closed under a binary operation $\cdot$ that has the following property: for each pair of elements a, b, there exists unique elements c_L and c_R such that $c_L \cdot a = b$ and $a \cdot c_R = b$. In contrast to groups, a quasigroup is not necessarily associative and does not need to have an identity. The Quasigroup Isomorphism problem QGroupIso is defined as GroupIso but the input structures are multiplication tables of quasigroups, also called Latin squares. GroupIso is trivially reducible to QGroupIso but a reduction in the other direction is not known. The complexity of both problems has been studied for more than three decades. Groups and quasigroups of order n have generator sets of size bounded by $\log n$. Because of this fact an isomorphism algorithm for GroupIso or QGroupIso running in time $n^{\log n + O(1)}$ can be obtained by computing a generator set of size $\log n$ in G_1, mapping this set in every possible way to a set of elements in G_2 and

[1] supported by a NSERC postdoctoral fellowship and research grants of Prof. Toniann Pitassi.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 317–326

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

checking whether by extending the mapping to all the (quasi)group elements following the multiplication tables of G_1 and G_2, an isomorphism is defined. This algorithm is attributed to Tarjan in [Mil78]. A stronger result showing that GroupIso can be solved in space $O(\log^2 n)$ was given in [LSZ76]. The same result for the case of quasigroups was obtained later by Wolf in [Wol94].

In spite of these facts, no deterministic polynomial time algorithm for these problems is known although they seem far from being NP-complete[1]. The status of the problems is similar to that of the better known Graph Isomorphism problem (GI). It is known that QGroupIso is AC^0 reducible to GI [Mil78], but the second one seems to be a harder problem. In this paper we prove this intuition by showing without assumptions that an AC^0 reduction in the other direction is not possible. This is done in two steps: first we improve the existing upper bound for QGroupIso to a class of polynomial size nondeterministic circuits of $O(\log\log n)$ depth (Section 3). Then in Section 4 we show that this circuit class cannot compute the Parity function. It follows that GroupIso and QGroupIso cannot be hard under AC^0 reductions for any class that is powerful enough to compute Parity, like NC^1 or L. This contrasts with the hardness properties of GI [Tor04, Tor10]. It also implies that GI cannot be AC^0 reducible to GroupIso or to QGroupIso .

The upper bound is based on the bounded nondeterminism properties of the problems. Observe that Tarjan's algorithm can in fact be converted into a polynomial time nondeterministic procedure for QGroupIso that uses only $\log^2 n$ nondeterministic bits, by guessing the mapping from the generator set in G_1 to G_2 instead of testing all possible 1-1 mappings, and then extend this partial map to the whole quasigroup. This observation is mentioned explicitly in [PY96, Wol94]. Papadimitriou and Yannakakis [PY96] show that the quasigroup isomorphism problem is in $\beta_2\mathrm{P}$, a restricted version of NP, where on input of length n, a polynomial time bounded Turing machine has access to $O(\log^2 n)$ non-deterministic bits (more detail is given in the preliminaries section). In [AT06] some evidence is given indicating that QGroupIso is probably not complete for $\beta_2\mathrm{P}$. Wolf [Wol94] improved the nondeterministic complexity of the problem by showing that QGroupIso $\in \beta_2\mathrm{NC}^2$ the class of problems computed by NC^2 circuits having additionally $O(\log^2 n)$ non-deterministic bits on inputs of size n. As in the $\beta_2\mathrm{P}$ upper bound, the circuit can guess the generators of both quasigroups as well as a bijection between both generator sets. Wolf shows that checking whether this partial bijection can be extended to an isomorphism, can be done by an NC^2 circuit. We improve this upper bound to $\beta_2\mathrm{FOLL}$, the class of problems computable by (uniform) families of polynomial size unbounded fan-in circuits with $O(\log\log n)$ depth and $O(\log^2 n)$ nondeterministic bits, where n is the number of quasigroup elements. The proof of this result is based on a special kind of generating sequences for the quasigroups called cube generating sequences. The cube generating sequences provide a representation for the structures that allow very quick isomorphism tests. Erdős and Rényi showed that groups have many generating sequences of this kind. We extend in Section 3 their result to the more general case of quasigroups.

The lower bound for $\beta_2\mathrm{FOLL}$ circuits for the Parity function is proved in Section 4, by first showing that computation by a few non-deterministic bits implies the existence of a polynomial size deterministic circuit of depth $O(\log\log n)$ that approximates Parity non-trivially. The argument is completed by a routine application of the decision-tree version of the Switching Lemma due to Razborov [Raz93] that rules out such approximations of Parity.

[1] In fact, we show in this paper, GroupIso and QGroupIso are not NP-complete under AC^0 reductions.

2 Preliminaries

2.1 Quasigroups

Given a set of elements from a quasigroup, a parenthesization specifies the sequence in which to multiply the elements. A parenthesization can be represented as a binary tree with the quasigroup elements at the leaves. The depth of a parenthesization is the depth of the binary tree representing it. For a quasigroup G and a set of elements $g_1, \ldots, g_l \in G^l$ and a parenthesization P we denote by $P(g_1, \ldots, g_l)$ the result of the multiplication of the elements according to P. For our results we need the following elementary fact.

▶ **Fact 1.** *Let P be any correct parenthesization for the multiplication of l elements in G. Then for every $i \in \{1, \ldots, l\}$ for every $b \in G$ and every fixed choice of elements $g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_l \in G^{l-1}$ there is a unique element $g_i \in G$ such that $P(g_1, \ldots, g_{i-1}, g_i, g_{i+1}, \ldots, g_l) = b$.*

Proof. By induction on l. For the base case $l = 2$ the quasigroup axioms imply the result. For $l > 2$ we consider the binary tree representing the parenthesization. We search for a value of g_i such that the equation $P(g_1, \ldots, g_{i-1}, g_i, g_{i+1}, \ldots, g_l) = b$ holds. The value of one of the successors of the root is determined by the values of $g_1, \ldots, g_{i-1}, g_{i+1}, \ldots, g_l$. W.l.o.g. let this be the left successor and denote its multiplication value by c. The value of the other successor must then be equal to d, the unique element in G with $c \cdot d = b$. By induction hypothesis there is a unique value for q_i such that the multiplication of the right subtree equals d. ◀

2.2 Complexity Classes

For the standard complexity classes used in this paper, like L or the circuit classes AC^i or NC^i we refer the reader to the standard books in complexity theory.

The complexity class FOLL, or $\mathsf{FO}(\log \log n)$, was introduced in [BKLM00] in order to characterize the complexity of the group membership problem. FOLL is the class of problems solvable by uniform polynomial size circuit families of unbounded fan-in and depth $O(\log \log n)$. Since the Parity function is not in FOLL, no problem in FOLL can be complete under AC^0-reductions for any class containing Parity, such as NC^1 or L. Currently AC^1 is the best upper bound for FOLL and the class is not known to be contained even in NL.

For a circuit class C, $\beta_k C$ is the class of languages recognized by a (uniform) family of C circuits with n input bits and $O(\log^k n)$ nondeterministic bits. We say that such a nondeterministic circuit accepts a string x if for some choice of the nondeterministic bits the circuit with input x outputs a one. Classes of bounded nondeterminism have appeared in different forms in the literature [KF84, DT90, PY96, GLM96]. As we show in this paper, the circuit setting is well suited to argue about these classes.

3 Nondeterministic Circuit Complexity of QGroupIso

We show in this section that QGroupIso can be solved by a uniform family of nondeterministic FOLL circuits with $O(\log^2 n)$ nondeterministic bits: QGroupIso $\in \beta_2$FOLL. This result improves a series of upper bounds of this kind for the problem: Papadimitriou and Yannakakis showed in [PY96] that QGroupIso $\in \beta_2$P this was improved to $\beta_2\mathrm{NC}^2$ by Wolf [Wol94] and more recently by Wagner to $\beta_2\mathrm{SAC}^1$ [Wag10].

In our proof the nondeterministic bits of the circuits are used in order to guess a special kind of generator sequence for both quasigroups. We call these generators cube generating sequences.

▶ **Definition 2.** A sequence of group elements $g = (g_0, g_1, \ldots, g_k)$ together with a parenthesization P for k elements is a *cube generating sequence* for quasigroup G if

$$G = \{P(g_0, g_1^{\epsilon_1}, \ldots, g_k^{\epsilon_k}) \mid \epsilon_i \in \{0,1\}\}$$

The set $\{P(g_0, g_1^{\epsilon_1}, \ldots, g_k^{\epsilon_k}) \mid \epsilon_i \in \{0,1\}\}$ is the *cube* $Cube(g, P)$ generated by the sequence g and the parenthesization P.

In a cube generating sequence, the generators are given in a fixed order. Erdős and Renyi [ER65] proved that every group with n elements has cube generating sequences of size $O(\log n)$. As a matter of fact there are many such short sequences. In the case of groups we do not need to talk about parenthesizations since the operation is associative.

▶ **Theorem 3.** *[ER65] Let G be a finite group with n elements. For any $\delta > 0$ the probability that a sequence of group elements of size $k \geq \log n + 2\log\frac{1}{\delta} + \log\log n + 5$ selected uniformly at random is a cube generating sequence for G, is $> 1 - \delta$.*

This result can be adapted to work also for quasigroups. For our purposes a simpler existential version of the result suffices. However we need to make sure that the multiplications of the generators can be be performed very fast in parallel and therefore we need a short cube generating sequence with shallow parenthesization.

▶ **Theorem 4.** *For a finite quasigroup G with n elements, there exists a cube generating sequence g for G, together with a parenthesization P such that g has $O(\log n)$ elements and P has depth $O(\log\log n)$.*

Proof. Let G be a quasigroup with n elements and for $k > 0$ let P be any fixed parenthesization of $k+1$ elements. Let $g_0, \ldots, g_k$ be $k+1$ elements chosen in G uniformly at random and independently of each other. For $b \in G$ let $V_k(b)$ be the number of representations of b of the form $b = P(g_0, g_1^{\epsilon_1}, \ldots, g_k^{\epsilon_k})$ with $\epsilon_i \in \{0,1\}$. For succinctness for $\epsilon = (\epsilon_1, \ldots, \epsilon_k) \in \{0,1\}^k$ and $g = (g_0, \ldots, g_k) \in G^{k+1}$ we represent $P(g_0, g_1^{\epsilon_1}, \ldots, g_k^{\epsilon_k})$ by $P(g^\epsilon)$ (or even g^ϵ when the parenthesization is clear).

For each $b \in G$, $V_k(b)$ is a random variable. We estimate its expectation and its variance. For a random sequence $g = (g_0, \ldots, g_k) \in G^{k+1}$ consider the indicator variable

$$X_\epsilon(b) = \begin{cases} 1 & \text{if } g^\epsilon = b \\ 0 & \text{otherwise.} \end{cases}$$

For random g, $\Pr[X_\epsilon(b) = 1] = \frac{1}{n}$. This is because $X_\epsilon(b) = 1$ if and only if $g^\epsilon = b$ and this is true exactly when g_0 is equal to the unique element $x \in G$ satisfying the equation $b = P(x, g_1^{\epsilon_1}, \ldots, g_k^{\epsilon_k})$ (Fact 1). Since g_0 is chosen uniformly at random this probability is $\frac{1}{n}$. It follows:

$$E[V_k(b)] = E\left[\sum_{\epsilon \in \{0,1\}^k} X_\epsilon(b)\right] = \sum_{\epsilon \in \{0,1\}^k} E[X_\epsilon(b)] = \frac{2^k}{n}.$$

For calculating the variance we observe that the random variables $X_\epsilon(b)$ are pairwise independent. For $\epsilon \neq \epsilon' \in \{0,1\}^k$ and for a random $g \in G^{k+1}$ and fixed $b \in G$ we estimate the probability $\Pr[g^\epsilon = g^{\epsilon'} = b]$. We can suppose there is a position i with $\epsilon_i = 1$ and $\epsilon'_i = 0$. $g^{\epsilon'} = b$ if and only if g_0 is equal to the unique element $x \in G$ satisfying the equation $b = P(x, g_1^{\epsilon'_1}, \ldots, g_k^{\epsilon'_k})$. If this holds then $g^\epsilon = b$ if and only if g_i is equal to the unique

element $y \in G$ satisfying $b = P(x, g_1^{\epsilon_1}, \ldots, g_{i-1}^{\epsilon_{i-1}}, y, g_{i+1}^{\epsilon_{i+1}}, \ldots, g_k^{\epsilon_k})$. Since g_0 and g_i are chosen independently, the probability that these two facts hold is then $\frac{1}{n^2}$. Now we can estimate the variance of $V_k(b)$. Since $V_k(b)$ is the sum of pairwise independent random variables its variance is the sum of the the variances of the summands. Therefore:

$$Var[V_k(b)] = Var\left[\sum_{\epsilon \in \{0,1,\}^k} X_\epsilon(b)\right] = \sum_{\epsilon \in \{0,1\}^k} Var[X_\epsilon(b)] = 2^k\left(\frac{1}{n} - \frac{1}{n^2}\right) < \frac{2^k}{n}$$

Let N_k be the number of elements in G not having any representation in the cube generated by a random sequence g of size k+1. We show next that $E[N_k] \leq \frac{n^2}{2^k}$. For this we need to estimate the probability that for an element $b \in G$, $V_k(b) = 0$.

$$\begin{aligned} \Pr[V_k(b) = 0] &\leq \Pr\left[\left|V_k(b) - \frac{2^k}{n}\right| \geq \frac{2^k}{n}\right] \\ &\leq \frac{\mathrm{Var}[V_k(b)]n^2}{2^{2k}} < \frac{n}{2^k} \end{aligned}$$

The second step follows by Chebyshev's inequality. We can now estimate the expectation for N_k.

$$E[N_k] = \sum_{b \in G} \Pr[V_k(b) = 0] \leq \frac{n^2}{2^k}.$$

Considering $k = \lceil 2 \log n \rceil + 1$ we have $E[N_k] < 1$, which means that there must be a sequence g of size $k+1$ that represents all the elements in G. Since this works for any parenthesization we can fix P to be a balanced binary tree with $k+1$ leaves and therefore depth $O(\log \log n)$.

◀

Observe that for a quasigroup G, a fixed k and a fixed parenthesization P, the family of functions obtained by choosing a sequence g of $k+1$ elements in G uniformly at random and mapping $\epsilon \in \{0,1\}^k$ to $g^\epsilon \in G$ (with parenthesization P) is in fact a 2-universal family of hash functions. As it can be seen in our previous proof, the argument does not need fully independence while choosing the elements in G, but just pairwise independence. As a consequence it is possible to obtain small cube generating sets for G deterministically. However this would not bring any advantage to our nondeterministic algorithm, since $O(\log^2 n)$ nondeterministic bits are needed to guess the cube generating set of the second input structure in a way that the isomorphism can be extended to all the elements in the canonical way.

We can now prove our upper bound for QGroupIso .

▶ **Theorem 5.** *The Quasigroup Isomorphism problem is in* β_2FOLL.

Proof. Let G, H be two quasigroups given as multiplication tables let $g = (g_1, \ldots, g_k)$ and $h = (h_1, \ldots, h_k)$ be generating sequences of the same length, and P be a balanced parenthesization with $G = \mathrm{Cube}(g, P)$ and $H = \mathrm{Cube}(h, P)$.

If we can prove that the function that maps g_i to h_i for $i \in \{1, \ldots, k\}$ can be extended to an isomorphism between G and G' then clearly both quasigroups are isomorphic. This is true if and only if for every $\epsilon, \epsilon', \epsilon'' \in \{0,1\}^k$ $g^\epsilon = g^{\epsilon'} g^{\epsilon''}$ if and only if $h^\epsilon = h^{\epsilon'} h^{\epsilon''}$. On the other hand if the quasigroups are not isomorphic, the function mapping g_i to h_i would not pass the mentioned isomorphism test.

This is the basis for the upper bound. $O(\log^2 n)$ nondeterministic bits in the circuit are used for guessing the cube generating sequences for G and H in the right order. The isomorphism tests can be done then in the depth of the multiplications which is the depth of the parenthesization P, that is $O(\log \log n)$.

---

input: Quasigroups G, H on elements in $\{1, \ldots, n\}$ given as multiplication tables, cube generating sequences $g = (g_0, g_1, \ldots, g_k)$ for G and $h = (h_0, h_1, \ldots, h_k)$ for H with balanced parenthesization P.

1: { test $G = Cube(g, P)$ and $H = Cube(h, P)$}
2: **for all** $a, b \in \{1, \ldots, n\}$
3: **for all** $(\epsilon_1, \ldots, \epsilon_k) \in \{0,1\}^k$
4: check whether $a = g_0 g_1^{\epsilon_1} \ldots g_k^{\epsilon_k}$ and $b = h_0 h_1^{\epsilon_1} \ldots h_k^{\epsilon_k}$
5: **if** a or b was not generated by any ϵ then reject and halt.
6: { isomorphism test }
7: **for all** $(\epsilon_1, \ldots, \epsilon_k) \in \{0,1\}^k$
8: **for all** $(\eta_1, \ldots, \eta_k) \in \{0,1\}^k$
9: $c \leftarrow g_0 g_1^{\epsilon_1} \ldots g_k^{\epsilon_k}, d \leftarrow g_0 g_1^{\eta_1} \ldots g_k^{\eta_k}$
10: $c' \leftarrow h_0 h_1^{\epsilon_1} \ldots h_k^{\epsilon_k}, d' \leftarrow h_0 h_1^{\eta_1} \ldots h_k^{\eta_k}$
11: **for all** $(\nu_1, \ldots, \nu_k) \in \{0,1\}^k$
12: **if** $cd = g_0 g_1^{\nu_1} \ldots g_k^{\nu_k} \leftrightarrow c'd' \neq h_0 h_1^{\nu_1} \ldots h_k^{\nu_k}$ **then** halt and reject.
13: halt and accept.

Since $k \in O(\log n)$, the number of performed ϵ-tests is bounded by a polynomial. Because of the parenthesization P, every multiplication $g = g_1^{\epsilon_1} \ldots g_k^{\epsilon_k}$ can be computed by a sub-circuit of depth $O(\log \log n)$ with unbounded fan-in. Each sub-circuit is organized as a pyramid. At the bottom level it uses the multiplication tables to multiply pairs of elements $g_i^{\epsilon_i} g_{i+1}^{\epsilon_{i+1}}$. At the next level it multiplies pairs of results of the previous level, and so on. The depth of the sub-circuits is bounded by $O(\log \log n)$ since $k \in O(\log n)$. ◀

The upper bound that we get for groups is the same one. For concrete group families it is possible to get better bounds. We include as example the case of Abelian groups.

On the Complexity of Abelian Group Isomorphism

We consider here the easier case when the input structures are Abelian groups.

Clearly, testing the property whether G is Abelian can be done in AC^0 by simply testing whether $a \cdot b = b \cdot a$ holds for all elements a, b in parallel. The isomorphism test is based on the following well known fact.

▶ **Fact 6.** *Two finite Abelian groups G and H with $|G| = |H| = n$ are isomorphic iff the number of elements of order m in G and H is the same, for all $1 \leq m \leq n$.*

A proof of this fact can be found for example in [Hal59]. The order of an element a is the smallest integer $i \geq 0$ such that $a^i = e$. Hence, an isomorphism test simply computes the orders for all elements using the power predicate. Barrington et.al. [BKLM00] considered the complexity of the power predicate on Abelian groups.

▶ **Lemma 7.** *([BKLM00]) Let G be a finite group given by its multiplication table. For all elements a and b in G and all $i \leq n$, the predicate $b = a^i$ can be computed in* FOLL $\cap$ L.

In the isomorphism test, an FOLL circuit computes the order of all group elements. This is a set of numbers in arbitrary order.

Given two multisets of numbers, the problem of pairwise comparing them is not in AC^0, since the Majority function reduces to this problem. It is known that Sorting, i.e. arranging n n-bit numbers in ascending order, is in TC^0. This suffices for an isomorphism test. When given two sorted multisets of numbers, say $e_1 \leq \cdots \leq e_n$ and $e'_1 \leq \cdots \leq e'_n$, it can be tested in AC^0 whether they coincide. We conclude:

▶ **Theorem 8.** *The Abelian Cayley-group isomorphism problem is in* $TC^0(\text{FOLL})$*, and in* L.

4 Computing Parity by Shallow Circuits with Limited Non-Determinism

We prove in this section that FOLL circuits (in fact polynomial size circuits of depth $O\big((\log\log n)^k\big)$) cannot compute the Parity function even with the help of poly-logarithmic many nondeterministic bits.

▶ **Theorem 9.** *Let C be a circuit of polynomial size and depth $O\big((\log\log n)^k\big)$, with access to $O\big((\log n)^\ell\big)$-many non-deterministic bits, where k, ℓ are arbitrary constant numbers. Then C cannot compute the Parity function.*

Proof. Let C be computing Parity and have depth d. Then for every possible setting of the nondeterministic bits C outputs zero for inputs of even parity. On the other hand, by averaging, there exists at least one setting θ of the non-deterministic bits for which C outputs 1 on at least $\frac{2^{n-1}}{2^{(\log n)^\ell}}$ many inputs of odd parity. Thus, the deterministic circuit C_θ obtained from C by fixing its non-deterministic bits to θ approximates Parity well, i.e.

$$\Pr_x\big[C_\theta(x) = \text{Parity}(x)\big] \geq \frac{1}{2} + \frac{1}{2\cdot 2^{O((\log n)^\ell)}}.$$

However, C_θ has the same size and depth as C. The proof gets completed by showing below, via Theorem 12, that such approximations to Parity are impossible.

◀

In order to prove the desired inapproximability results, we use a version of the Switching Lemma. Switching Lemmas were developed in a series of works by [FSS81, Ajtai83, Yao85, Cai86, Has87] for proving lower bounds on the size of constant-depth circuits computing Parity. We recall the following decision-tree version, due to Razborov [Raz93]. Let R_n^m be the space of all restrictions on n variables that leaves precisely m of them free. For any restriction ρ, we denote by f_ρ the boolean function induced from f on variables left free by ρ.

▶ **Lemma 10** (Switching Lemma, Razborov)**.** *Let f be a CNF (or DNF) formula with clause width t on n variables. Let ρ be a random restriction in R_n^m. Then, there exists a constant $\gamma > 0$ such that the probability of f_ρ not having a decision tree of height at most s is less than $\big(\frac{\gamma m t}{n}\big)^s$.*

An immediate consequence of this lemma is the following corollary:

▶ **Corollary 11.** *Let f be a function computed by a circuit of size S and depth d. Let $m = n/\big((2\gamma)^d (n^{1/(2d)})^{d-1}\big)$. Then*

$$\Pr_{\rho\in R_n^m}\Big[h(f_\rho) > n^{1/2d}\Big] \leq S\cdot\frac{1}{2^{\Omega\left(n^{1/2d}\right)}}$$

where $h(f_\rho)$ denotes the height of the best decision tree for f_ρ.

Proof. This can be shown by a simple inductive argument using the Switching Lemma. Assume, as our inductive hypothesis, the following: let $i \geq 2$ and $n_i = n/\big((2\gamma)^i (n^{1/(2d)})^{i-1}\big)$. Let G_i be the set of gates in the ith layer of C and let S_i be the number. Further, let $S_{\leq i} = \sum_{j=1}^{i} S_j$. Our inductive hypothesis is the following:

$$\Pr_{\rho \in R_n^{n_i}} \left[\exists g \in G_i \,:\, h\big(f_\rho^g\big) \,>\, n^{1/2d} \right] \leq S_{\leq i} \cdot \frac{1}{2^{n^{1/2d}}},$$

where f^g is the function computed at gate g. Now, if the ith layer of the circuit has AND (OR) gates then one can assume w.l.o.g that $i+1$th layer has OR (AND) gates. In this case, assuming that each f_ρ^g has a decision tree of height at most $n^{1/2d}$, we represent f_ρ^g as a DNF of width at most $n^{1/2d}$ by using the small height decision tree. This collapses layers i and $i+1$ and hence the output of every gate at layer $i+1$ is a DNF of width $n^{1/2d}$ under the restriction ρ. We apply the Switching Lemma to each such DNF where $n = n_i$, $m = n_{i+1}$ and $t = n^{1/2d}$. Clearly, the probability that any fixed such DNF under the next round of restriction fails to have a decision tree of height at most $n^{1/2d}$ is at most $2^{-n^{1/2d}}$. Applying the union bound to S_{i+1} such DNF's (one for each gate at layer $i+1$) immediately completes the induction.

◀

Applying the above, we get the following inapproximability result (which is possibly implicit in work of Cai[Cai86]):

▶ **Theorem 12.** *Let C be any polynomial size circuit of depth d. Then,*

$$\Pr_x \big[C(x) = \mathrm{Parity}(x)\big] \leq \frac{1}{2} + \frac{1}{2^{\Omega\left(n^{1/2d}\right)}}.$$

Proof. Applying Corollary 11, we see that if we pick a random restriction that leaves m variables free, where $m = n/\big((2\gamma)^d (n^{1/(2d)})^{d-1}\big)$ with probability at least $1 - \mathrm{Size}(C) \cdot 2^{-n^{1/2d}}$, the circuit will have a decision tree of height at most $n^{1/2d}$. Hence, with that much probability the number of free variables m is more than the height of the decision tree. For each such restriction, the restricted circuit computes the right answer (which is either Parity or its complement, on the m free variables) with probability exactly a half. Hence, even assuming that for all other restrictions we get perfect correlation,

$$\Pr_x \big[C(x) = \mathrm{Parity}(x)\big] \leq \frac{1}{2} + \Pr_{\rho \in R_n^m} \left[h\big(C_\rho\big) > n^{1/2d} \right] \leq \frac{1}{2} + \mathrm{Size}\big(C\big) \cdot \frac{1}{2^{n^{1/2d}}}.$$

The proof is completed by observing that the size of the circuit, denoted by $\mathrm{Size}\big(C\big)$, by assumption is polynomial.

◀

5 Discussion

Although no polynomial time algorithms for GroupIso or QGroupIso are known, we have shown in this paper that the problems are not hard enough to encode the Parity function. Therefore these problems cannot be hard under AC^0 reductions for any complexity class containing Parity, like L or NC^1. This contrasts sharply with the hardness properties of other isomorphism problems like Graph Isomorphism. In fact, our research started originally trying to prove that QGroupIso is hard for NC^1. At first sight it looks as if the difficulty in encoding

the Parity function comes from the very structured way in which the input information is presented in the the multiplication tables. The way of proving the result, however was to show that the computation of QGroupIso can be divided in two faces, a first bounded nondeterministic part and a very efficient checking part. We then gave an upper bound for the checking part in terms of circuits with very restricted depth and showed that these circuits cannot compute Parity even with the help of poly-log many nondeterministic bits. We observe that this proof technique does not have anything to do with isomorphism problems and can be applied to other problems whose computation have similar bounded guessing and checking parts. For example the classes LOGNP_0 and LOGSNP_0 from [PY96] would fall in $\beta_2\mathrm{AC}^0$ in our setting. The results in this paper imply that the problems in these classes cannot be AC^0 hard for Parity. Observe that for example the problem LOGCLIQUE, deciding if a given graph with n vertices has a clique of size at least $\log n$ falls into this category. We find this surprising. It would be interesting to study, maybe with other techniques, the existence of longer hierarchies of natural problems defining different AC^0 degrees.

So far all the upper bounds known for GroupIso hold also for QGroupIso . The question of whether the problems are equivalent under some reduction remains open.

Acknowledgments. We thank the anonymous referees for helpful comments on the manuscript.

References

Ajtai83 Miklós Ajtai. Σ^1_1-formulae on finite structures. In *Annals of Pure and Applied Logic*, 24:1–48, 1983.

AT06 V. Arvind and Jacobo Torán. The complexity of quasigroup isomorphism and the minimum generating set problem. In Tetsuo Asano, editor, *International Symposium on Algorithms and Computation (ISAAC)*, volume 4288 of *Lecture Notes in Computer Science*, pages 233–242. Springer, 2006.

BKLM00 David Mix Barrington, Peter Kadau, Klaus-Jörn Lange, and Pierre McKenzie. On the complexity of some problems on groups input as multiplication tables. In *Proceedings of the 15th Annual IEEE Conference on Computational Complexity (COCO)*, page 62, Washington, DC, USA, 2000. IEEE Computer Society.

Cai86 Jin-yi Cai. With probability one, a random oracle separates PSPACE from the polynomial-time hierarchy. In *Proceedings of the Eighteenth Annual ACM Symposium on Theory of Computing (STOC)* 38(1), 21–29, 1986.

DT90 Josep Díaz and Jacobo Torán. Classes of Bounded Nondeterminism. *Mathematical Systems Theory* 23(1): 21–32, 1990.

FSS81 Merrick L. Furst, James B. Saxe and Michael Sipser. Parity, circuits and the polynomial-time hierarchy. In *22nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 260–270, 1981.

ER65 Paul Erdős and Alfred Rényi. Probabilistic methods in group theory. *Journal d'Analyse Mathématique*, 14:127–138, 1965.

GLM96 Judy Goldsmith, Matthew A. Levy and Martin Mundhenk. Limited nondeterminism. *SIGACT News* 27(2): 20–29, 1996.

Hal59 Marshall Hall. *The theory of groups.* Macmillan, New York, 1959.

Has87 John Håstad *Computational limitations of small-depth circuits.* MIT Press, 1987.

KF84 Chandra Kintala and Patrick Fisher. Refining nondeterminism in relativized complexity classes. *SIAM Journal on Computing* 13:329–337, 1984.

LSZ76 Richard J. Lipton, Lawrence Snyder, and Yechezkel Zalcstein. The complexity of word and isomorphism problems for finite groups. Technical report, John Hopkins, 1976.

Mil78 Gary L. Miller. On the $n^{log n}$ isomorphism technique. In *ACM Symposium on Theory of Computing (STOC)*, 1978.

PY96 Christos H. Papadimitriou and Mihalis Yannakakis. On limited nondeterminism and the complexity of the VC dimension. *Journal of Computer and System Sciences*, 53:161–170, 1996.

Raz93 Alexander A. Razborov. An equivalence between second order bounded domain bounded arithmetic and first order bounded arithmetic. In P. Clote and J. Krajíček, editors, *Arithmetic, Proof Theory and Computational Complexity*, Oxford University Press (1993), 247–277.

Tor04 Jacobo Torán. On the hardness of Graph Isomorphism. *SIAM Journal on Computing* 33(5): 1093–1108, 2004.

Tor10 Jacobo Torán. Reductions to Graph Isomorphism. *Theory of Computing Systems* 47(1): 288–299, 2010.

Wag10 Fabian Wagner. On the complexity of isomorphism testing for restricted classes of graphs. Ph.D. Thesis. Technical Report VTS-ID/7264, Institutional Repository of University of Ulm, 2010.

Wol94 Marty J. Wolf. Nondeterministic circuits, space complexity and quasigroups. *Theoretical Computer Science (TCS)*, 125:295–313, 1994.

Yao85 Andrew C.C. Yao. Separating the polynomial hierarchy by oracles: Part I. In *26th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1–10, 1985.

Colored Hypergraph Isomorphism is Fixed Parameter Tractable

V. Arvind[1], Bireswar Das[2], Johannes Köbler[3], and Seinosuke Toda[4]

1 **The Institute of Mathematical Sciences, Chennai 600 113, India**
`arvind@imsc.res.in`
2 **Indian Institute of Technology, Gandhinagar, India**
`bireswar@iitgn.ac.in`
3 **Institut für Informatik, Humboldt Universität zu Berlin, Germany**
`koebler@informatik.hu-berlin.de`
4 **Nihon University, Tokyo, Japan**
`toda@cssa.chs.nihon-u.ac.jp`

Abstract

We describe a fixed parameter tractable (fpt) algorithm for Colored Hypergraph Isomorphism which has running time $2^{O(b)}N^{O(1)}$, where the parameter b is the maximum size of the color classes of the given hypergraphs and N is the input size. We also describe fpt algorithms for certain permutation group problems that are used as subroutines in our algorithm.

Keywords and phrases Fixed parameter tractability, fpt algorithms, graph isomorphism, computational complexity.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.327

1 Introduction

A *hypergraph* is an ordered pair $X = (V, E)$ where V is the vertex set and $E \subseteq 2^V$ is the edge set. Two hypergraphs $X = (V, E)$ and $X' = (V', E')$ are said to be *isomorphic*, denoted $X \cong X'$, if there is a bijection $\varphi : V \to V'$ such that for all $e = \{u_1, \cdots, u_l\} \subseteq V$, $e \in E$ if and only if $\varphi(e) = \{\varphi(u_1), \cdots, \varphi(u_l)\} \in E'$. Given two hypergraphs X and X', the decision problem Hypergraph Isomorphism (HI) asks whether $X \cong X'$. Graph Isomorphism (GI) is obviously polynomial-time reducible to HI. Conversely, HI is also known to be polynomial-time reducible to GI: Given a pair of hypergraphs $X = (V, E)$ and $X' = (V', E')$ as instance for HI, the reduced instance of GI consists of two corresponding bipartite graphs Y and Y' defined as follows. The graph Y has vertex set $V \uplus E$ and edge set $E(Y) = \{\{v, e\} \mid v \in V, e \in E \text{ and } v \in e\}$, and Y' is defined similarly. Here, $C \uplus D$ denotes the disjoint union of the sets C and D. It is easy to verify that $Y \cong Y'$ if and only if $X \cong X'$ assuming that V can be mapped only to V' and E can be mapped only to E'. This latter condition is easy to enforce.

However, since the above reduction blows up the size of the vertex set in the bipartite encoding, the Zemlyachenko-Luks-Babai graph isomorphism algorithm [3, 5, 6, 25] that runs in time $c^{\sqrt{n \log n}}$, where n is the size of the vertex set of the graph, does not yield a similar algorithm for hypergraph isomorphism. We note here that the best known hypergraph isomorphism test due to Luks [16] has running time c^n. Recently, Babai and Codenotti [4] have shown a $2^{\tilde{O}(k^2\sqrt{n})}$ isomorphism testing algorithm for hypergraphs with hyperedges of size bounded by k.

Motivated by this situation, we explore the same algorithmic problem for bounded color class hypergraphs. The input to Colored Hypergraph Isomorphism (CHI) is a pair

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 327–337

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

of hypergraphs $X = (V, E)$ and $X' = (V', E')$ together with partitions $V = C_1 \uplus \cdots \uplus C_k$ and $V' = C'_1 \uplus \cdots \uplus C'_k$ of their vertex sets into *color classes* C_i and C'_i, respectively. The problem is to decide if there is an isomorphism φ that preserves the colors (meaning that $v \in C_i \Leftrightarrow \varphi(v) \in C'_i$). Colored Graph Isomorphism (CGI) is the analogous problem where instead of hypergraphs we have graphs as inputs.

CGI with color classes of size bounded by a constant is the first special case of GI shown to be in polynomial time [2, 12] and which brought in the application of permutation group theory to the problem. In fact, Babai [2] and Furst, Hopcroft and Luks [12] even gave an fpt algorithm for CGI with running time $O(b!)n^{O(1)}$, where the parameter b is the maximum size of the color classes and n is the number of vertices of the input graphs. By using the halving technique as introduced in [5] (see also [16]), the running time can be improved to $2^{O(b)}n^{O(1)}$.

In [13] a *complexity-theoretic* study of some special cases of bounded color class Graph Isomorphism has been done in connection to logarithmic space-bounded complexity classes. This line of research is continued in [1], where special cases of bounded color class Graph Isomorphism as well as Hypergraph Isomorphism are studied from a complexity theory perspective.

In this paper our focus is on designing an efficient algorithm for CHI. Although HI is polynomial time many-one reducible to GI, the reduction we described above does not impose any bound on the size of the color classes of the bipartite graphs Y and Y'. More specifically, if the color classes of the hypergraphs X and X' have size at most b, then the vertices of the graphs Y and Y' that correspond to the edges of X and X' do not get partitioned into color classes of size bounded by any function of b. Thus, the fpt algorithm for CGI cannot be combined with the above reduction to get an fpt algorithm for CHI. Moreover, even if b is bounded by a constant (say 2), the color classes in the resulting bipartite graphs can have size exponential in n where n is the number of vertices and hence, this approach would not even give a polynomial time isomorphism algorithm for hypergraphs with color class bound 2.

However, an algorithm for CHI running in time $N^{O(b)}$ was shown in [19], where b bounds the size of the color classes of the given hypergraphs and N is the input size. Hence, if b is bounded by a constant, we have already a polynomial-time algorithm for CHI. This algorithm basically applies Luks's seminal result [15] showing that the set stabilizer problem with respect to a class of permutation groups Γ_d can be solved in time $n^{O(d)}$.

Parametrized complexity and isomorphism testing

Parametrized complexity is a fundamental strategy for coping with intractability. Pioneered by Downey and Fellows in [8], it is a flourishing area of research (see, e.g. the monographs [9, 11]). Fixed parameter tractability provides a notion of feasible computation less restrictive than polynomial time. It provides a theoretical basis for the design of new algorithms that are efficient and practically useful for small parameter values.

Parametrized complexity theory deals with the study and design of algorithms that have a running time of the form $f(b)n^{O(1)}$ where n is the input size, b is the parameter and f is a computable function. If a problem is solvable by such an algorithm it is called *fixed parameter tractable* (fpt).

Since no polynomial-time algorithm for GI is known, one approach is to design fpt isomorphism testing algorithms with respect to natural graph parameters. For example, the algorithm of Babai and Furst et al [2, 12] mentioned above is fpt with respect to the color class size. For isomorphism testing of graphs with eigenvalue multiplicity bounded by k,

Evdokimov and Ponomarenko have designed a highly nontrivial fpt algorithm with running time $k^{O(k)}n^{O(1)}$ [10].

Apart from this, fpt algorithms have also been designed with respect to the parameters tree-distance width [24] and the size of the simplicial components of the input graphs [23]. Very recently, it is shown in [14] that Graph Isomorphism for graphs with feedback vertex sets of size k is fixed parameter tractable, with k as the parameter.

On the other hand, if we use the maximum degree [15], or the treewidth [7], or the genus [18] of the input graphs as parameter b, the best known isomorphism testing algorithms have a worst-case running time bound $n^{O(b)}$. It is an interesting open question if GI has an fpt algorithm with respect to any of these three parameters.

Our result

In this paper we present an fpt algorithm for Colored Hypergraph Isomorphism that runs in time $2^{O(b)}N^{O(1)}$, where b is the maximum size of the color classes and N is the input size (which we can define as $N = mn$, where n is the number of vertices and m is the number of hyperedges).

Broadly speaking, our algorithm is a combination of divide and conquer with dynamic programming. We adapt ideas from [5, 16] which applies the halving technique in combination with dynamic programming. Luks [16] gives a $2^{O(n)}$ time algorithm for Hypergraph Isomorphism. Our algorithm can be seen as a generalization of Luks's result.

We use as subroutines fpt algorithms for certain permutation group problems (mainly, the coset intersection problem) parametrized by the size of the largest *color class* of the group. While the parametrized complexity of permutation group problems, for different parameters, is certainly interesting in its own right, it could also be applicable to GI. For example, an fpt algorithm for Set Transporter w.r.t. groups in Γ_d (with d as parameter) would result in an fpt algorithm for testing isomorphism of graphs of degree $\leq d$.

2 Preliminaries

In this section we recall some basic group theory. Let G be a finite group and let Ω be a finite nonempty set. The *action* of the group G on Ω is defined by a map $\alpha : \Omega \times G \to \Omega$ such that for all $x \in \Omega$, (i) $\alpha(x, id) = x$, i.e., the identity $id \in G$ fixes each $x \in \Omega$, and (ii) $\alpha(\alpha(x, g), h) = \alpha(x, gh)$ for all $g, h \in G$. We write x^g instead of $\alpha(x, g)$ when the group action is clear from the context.

For $x \in \Omega$, its *G-orbit* is the set $x^G = \{y | y \in X, y = x^g \text{ for some } g \in G\}$. When the group is clear from the context, we call x^G the *orbit* of x. Notice that the orbits form a partition of Ω.

We write $H \leq G$ when H is a subgroup of G. The *symmetric group* on a finite set Ω consisting of all permutations on Ω is denoted by $\mathrm{Sym}(\Omega)$. If $\Omega = [n] = \{1, \cdots, n\}$, we write S_n instead of $\mathrm{Sym}([n])$. A *finite permutation group* G is a subgroup of $\mathrm{Sym}(\Omega)$ for some finite set Ω.

The permutation group *generated* by a subset $S \subseteq \mathrm{Sym}(\Omega)$ is the smallest subgroup of $\mathrm{Sym}(\Omega)$ containing S and is denoted by $\langle S \rangle$. Each element of the group $\langle S \rangle$ is expressible as a product of elements of S.

The subgroup $G^{(i)}$ of $G \leq S_n$ that fixes each of $\{1, \ldots, i\}$ is called a *pointwise stabilizer* of G. These subgroups form a tower

$$G = G^{(0)} \geq G^{(1)} \geq G^{(2)} \geq \cdots \geq G^{(n-1)} = \{id\}.$$

We notice that by the orbit-stabilizer lemma, the index $[G^{(i-1)} : G^{(i)}]$ is at most n. For each i, let R_i be a set of complete and distinct coset representatives of $G^{(i)}$ in $G^{(i-1)}$. Then $\bigcup_{i=1}^{n-1} R_i$ generates G and is called a *strong generating set* for G. Given a permutation $\pi \in G$ it is easy to check if $\pi \in G^{(i)}$. It is also easy to check if two permutations $\pi, \sigma \in G^{(i)}$ are in the same coset of $G^{(i+1)}$ in $G^{(i)}$. We just have to test if $\pi^{-1}\sigma \in G^{(i+1)}$. These observations yield a polynomial-time algorithm [21, 22, 12] for computing a strong generating set of a permutation group G. This algorithm can also be used to test in polynomial time if $g \in S_n$ is in the group $\langle S \rangle \leq S_n$.

In some applications there is an efficient algorithm for testing membership in a subgroup H of G, where $G \leq S_n$ is given by a generating set but no generating set for H is given. By [21, 22, 12] we can efficiently compute a generating set for H provided that its index in G is polynomially bounded.

▶ **Theorem 1** (Schreier Generators). *Let $G = \langle S \rangle \leq S_n$ and $H \leq G$. Then for any set R of coset representatives of H in G, the set $B = \{r'xr^{-1} \mid r, r' \in R, x \in S\} \cap H$ generates H. The generators in B are called* Schreier generators.

The proof of Theorem 1 also provides an algorithm for computing a suitable set R of coset representatives by making $m^2|S|$ tests of membership in H, where $m = [G : H]$. Though the set B of Schreier generators for H can be of size polynomial in m, it is possible to convert it to a strong generating set for H of size $O(n^2)$ [21, 22, 12].

For a permutation $\pi \in \mathrm{Sym}(\Omega)$ and a subset $C \subseteq \Omega$ we use C^π to denote the set $\{x^\pi \mid x \in C\}$. For a set S of permutations, C is called *S-stable* if $C^\pi = C$ for all $\pi \in S$. For a permutation group $G \leq \mathrm{Sym}(\Omega)$, the *stabilizer subgroup* of G is defined as $G_C = \{\pi \in G \mid C^\pi = C\}$.

3 Permutation group problems

In this section we describe fpt algorithms for some permutation group problems with respect to the color class bound as parameter. These algorithms are useful subroutines for our main algorithm which will be described in the next section.

A permutation group $G \leq \mathrm{Sym}(\Omega)$ has *color class bound* b if Ω is a colored set partitioned into *color classes* $\Omega = C_1 \uplus \cdots \uplus C_k$ such that $|C_i| \leq b$ for each i and each C_i is G-stable. Equivalently, the maximum orbit size of G is bounded by b. Since the orbits of G can be computed in $|\Omega|^{O(1)}$ time (for G given by a generating set S), we can determine in $|\Omega|^{O(1)}$ time if G has color class bound b. We first consider the following parametrized version of the set transporter problem.

Set Transporter

Input: A generating set for a group $G \leq \mathrm{Sym}(\Omega)$, a permutation $z \in \mathrm{Sym}(\Omega)$, subsets $\Pi_1, \ldots, \Pi_k, \Pi'_1, \ldots, \Pi'_k \subseteq \Omega$ and a partition $\Omega = C_1 \uplus \cdots \uplus C_k$ such that for each i, C_i is G-stable and $\Pi_i, \Pi'_i \subseteq C_i$.

Parameter: $b = \max\{|C_1|, \cdots, |C_k|\}$.

Output: A description of $(Gz)_{\Pi_1,\ldots,\Pi_k \to \Pi'_1,\ldots,\Pi'_k} = \{x \in Gz \mid \Pi_i^x = \Pi'_i \text{ for } i = 1, \ldots, k\}$.

The simple fpt algorithm for Set Transporter works by solving the problem for the first color class C_1 by computing the subcoset $G_1 z_1$ of Gz that maps Π_1 to Π'_1, then computing the subcoset $G_2 z_2$ of $G_1 z_1$ that maps Π_2 to Π'_2 and so on until all the color classes are dealt with. The following lemma shows how to compute $G_i z_i$ from $G_{i-1} z_{i-1}$.

▶ **Lemma 2.** *There is an fpt algorithm running in time* $2^{O(b)}n^{O(1)}$ *that computes the subcoset* $H'y'$ *of the coset* Hy *that maps* Π_i *to* Π'_i*, where* $\Pi_i, \Pi'_i \subseteq C_i$*.*

Proof. Let $H_{\Pi_i} = \{x \in H \mid \Pi_i^x = \Pi_i\}$ be the subgroup of H that stabilizes Π_i. Let $|\Pi_i| = \ell$. Since C_i is H-stable the set Π_i^x is also a size ℓ subset of C_i. It follows that

$$[H : H_{\Pi_i}] \leq \binom{b}{\ell} \leq 2^b.$$

Also note that given $x \in H$, it only takes $O(n)$ time to check if $x \in H_{\Pi_i}$. Applying the algorithm given by Theorem 1 we can compute a set $R = \{\rho_1, \cdots, \rho_t\}$ of coset representatives of H_{Π_i} in H in time $2^{O(b)}n^{O(1)}$ together with a strong generating set S for H_{Π_i} of size at most n^2. Writing

$$Hy = H_{\Pi_i}\rho_1 y \uplus \cdots \uplus H_{\Pi_i}\rho_t y,$$

the algorithm picks the uniquely determined coset $H_{\Pi_i}\rho_i y$ that sends Π_i to Π'_i and outputs the pair $(S, \rho_i z)$ as a description of the coset $H_{\Pi_i}\rho_i y$. If none of the cosets $H_{\Pi_i}\rho_i z$ maps Π_i to Π'_i, the algorithm outputs the empty set. ◀

▶ **Theorem 3.** *There is an fpt algorithm for* Set Transporter *running in time* $2^{O(b)}n^{O(1)}$*, where* $b = \max\{|C_1|, \cdots, |C_k|\}$ *and* $n = |\Omega|$*.*

Proof. Let $G_0 = G$ and $z_0 = z$ and for $i = 1, \cdots, k$ use the algorithm of Lemma 2 to compute

$$G_i z_i = (G_{i-1} z_{i-1})_{\Pi_i \to \Pi'_i}.$$

Notice that for each $x \in G_k z_k$ we have $\Pi_i^x = \Pi'_i$ for $i = 1, \ldots, k$, implying that $G_k z_k = (Gz)_{\Pi_1,\ldots,\Pi_k \to \Pi'_1,\ldots,\Pi'_k}$.

Furthermore, each of the subgroups G_i stabilizes the sets C_j, $j = 1, \cdots, k$. Thus, Lemma 2 implies that we can compute $G_i z_i$ from $G_{i-1} z_{i-1}$ in time $2^{O(b)}n^{O(1)}$, implying that the overall running time is also $2^{O(b)}n^{O(1)}$. ◀

Next we consider the following parametrized version of the coset intersection problem.

Coset Intersection (Coset-Inter)

Input: Generating sets for groups $G, H \leq \mathrm{Sym}(\Omega)$, permutations $x, y \in \mathrm{Sym}(\Omega)$ and a partition $\Omega = C_1 \uplus \cdots \uplus C_k$ such that for each i, C_i is $G \cup H \cup \{x, y\}$-stable.
Parameter: $b = \max\{|C_1|, \cdots, |C_k|\}$.
Output: $Gx \cap Hy$.

Applying well-known techniques from [5] we will design an fpt algorithm for Coset-Inter. We will use this as a subroutine in the next section to solve Colored Hypergraph Isomorphism. Our fpt algorithm for Coset-Inter will require solving a subproblem which is a restricted version of the set stabilizer problem.

Restricted Set Stabilizer (RSS)

Input: A generating set for a group $L \leq \mathrm{Sym}(\Omega_1) \times \mathrm{Sym}(\Omega_2)$, a permutation $z \in \mathrm{Sym}(\Omega_1 \times \Omega_2)$ and subsets $\Pi, \Theta = \Phi \times \Psi \subseteq C \times D$, where $\Omega_1 = C \uplus U$, $\Omega_2 = D \uplus V$ and the two sets $C \times D$ and Θ are L-stable.
Parameter: $b = \max\{|C|, |D|\}$.
Output: $(Lz)_\Pi[\Theta] = \{x \in Lz \mid (\Pi \cap \Theta)^x = \Pi \cap \Theta^x\}$.

▶ **Lemma 4.** *There is an fpt algorithm for* RSS *running in time* $2^{O(b)}n^{O(1)}$*, where* $b = \max\{|C|, |D|\}$ *and* $n = |\Omega_1| + |\Omega_2|$.

Proof. We use ideas from [16, Proposition 3.1] where the author describes an algorithm for a version of the set transporter problem that can be easily adapted to solve RSS. These ideas were first applied in [5]. We only have to slightly modify Luks's proof to suit the parametrized setting.

We can assume that $|\Phi|$ and $|\Psi|$ are powers of 2 since otherwise we can add some points to Φ and Ψ (as well as to C and D) and let L act trivially on these points. This will increase the size of b and of the input only by a factor of 4. Further, these extra points can be easily removed from the algorithm's output.

Observe that since $L_\Theta = L$, we have $\Theta^x = \Theta^z$ for all $x \in Lz$. If $(Lz)_\Pi[\Theta]$ is not empty then for $x, y \in (Lz)_\Pi[\Theta]$ we have $(\Pi \cap \Theta)^x = \Pi \cap \Theta^z = (\Pi \cap \Theta)^y$ and hence $(Lz)_\Pi[\Theta]$ is a coset of $L_{\Pi\cap\Theta}$.

Clearly, if $|\Pi \cap \Theta| \neq |\Pi \cap \Theta^z|$ then $(Lz)_\Pi[\Theta]$ is empty. Next we consider the case that $|\Pi \cap \Theta| = |\Pi \cap \Theta^z| = 1$. Let $\Pi \cap \Theta = \{u\}$ and $\Pi \cap \Theta^z = \{v\}$. Let L_u be the stabilizer of the point u which can be computed using the Schreier-Sims method. Then we can express L as the disjoint union of cosets

$$L = L_u x_1 \uplus \cdots \uplus L_u x_t$$

and consequently Lz as $L_u x_1 z \uplus \cdots \uplus L_u x_t z$. Hence, it suffices to pick the uniquely determined coset $L_u x_i z$ that maps u to v (if there is any).

It remains to consider the case that $|\Pi \cap \Theta| = |\Pi \cap \Theta^z| > 1$. If $|\Phi| > 1$ we partition Φ in two subsets Φ_1 and Φ_2 of equal size and let $\Theta_1 = \Phi_1 \times \Psi$. Otherwise, $|\Psi_i| > 1$ and we partition Ψ in two subsets Ψ_1 and Ψ_2 of equal size and let $\Theta_1 = \Phi \times \Psi_1$. In both cases we let $\Theta_2 = \Theta \setminus \Theta_1$.

Let $k = \max\{|\Phi|, |\Psi|\}$ and let $M = L_{\Theta_1}$. Notice that $[L : M] \leq \binom{k}{k/2} \leq 2^b$, no matter which of the two sets Φ or Ψ we divide into two parts. Now we write L as the disjoint union of cosets

$$L = My_1 \uplus \cdots \uplus My_s$$

of M, yielding $Lz = My_1 z \uplus \cdots \uplus My_s z$. As mentioned in the preliminary section, this decomposition of Lz can be computed in time $2^{O(b)}n^{O(1)}$. Since M stabilizes Θ_1, we can use the equality

$$(My_i z)_\Pi[\Theta] = ((My_i z)_\Pi[\Theta_1])_\Pi[\Theta_2]$$

to set up the recursive calls. Finally we paste the answers to the subproblems $(My_i z)_\Pi[\Theta]$ together to get

$$(Lz)_\Pi[\Theta] = \cup_{i=1}^{t} (My_i z)_\Pi[\Theta].$$

It is easy to verify that the overall run-time of the algorithm is bounded by $2^{O(b)}poly(n)$. ◀

▶ **Theorem 5.** *There is an fpt algorithm for* Coset-Inter *running in time* $2^{O(b)}n^{O(1)}$*, where* $b = \max\{|C_1|, \cdots, |C_k|\}$ *and* $n = |\Omega|$.

Proof. Let $L = G \times H \leq \mathrm{Sym}(\Omega) \times \mathrm{Sym}(\Omega)$ and let $z = (x, y) \in \mathrm{Sym}(\Omega) \times \mathrm{Sym}(\Omega)$. Further, let $\Pi_i = \{(a, a) \mid a \in C_i\}$ and notice that $(Lz)_{\Pi_1,\ldots,\Pi_k} = \{x \in Lz \mid \Pi_i^x = \Pi_i \text{ for } i = 1, \ldots, k\}$ projected to the first (or second) coordinate is $Gx \cap Hy$. Hence, it suffices to prove the following claim.

Claim 6. *$(Lz)_{\Pi_1,\ldots,\Pi_k}$ is computable in time $2^{O(b)}n^{O(1)}$.*

We will repeatedly use Lemma 4 to solve the problem in time $2^{O(b)}n^{O(1)}$ as in the above claim. To start off we let $L_0z_0 = Lz$. Then we compute $L_iz_i = (L_{i-1}z_{i-1})_{\Pi_i}$ from $L_{i-1}z_{i-1}$ for $i = 1, \cdots, k$. We claim that for all i, $L_iz_i = (Lz)_{\Pi_1,\ldots,\Pi_i}$. This follows from the fact that $((Lz)_{\Pi_1,\ldots,\Pi_{i-1}})_{\Pi_i} = (Lz)_{\Pi_1,\ldots,\Pi_i}$. Thus at the end of the computation we have $L_kz_k = (Lz)_{\Pi_1,\ldots,\Pi_k}$. Furthermore, by Lemma 4 it follows that the time needed for computing L_iz_i from $L_{i-1}z_{i-1}$ is $2^{O(b)}n^{O(1)}$, implying that the overall running time is also $2^{O(b)}n^{O(1)}$. ◀

4 Fpt algorithms for Colored Hypergraph Isomorphism

In this section, we use a dynamic programming approach to design an fpt algorithm for finding the automorphism group $\mathrm{Aut}(X)$ (i.e., a set of generators for $\mathrm{Aut}(X)$) of a given hypergraph X which has running time $2^{O(b)}N^{O(1)}$.

▶ **Theorem 7.** *Let $X = (V, E)$ be a colored hypergraph of size N with $V = C_1 \uplus \cdots \uplus C_k$ where $|C_i| \leq b$ for all i. Given X as input there is an algorithm that computes $Aut(X)$ in time $2^{O(b)}N^{O(1)}$.*

Proof. The algorithm first partitions the hyperedges into different subsets that we call blocks. More formally, we say that two hyperedges $e_1, e_2 \in E$ are *i-equivalent* and write $e_1 \equiv_i e_2$, if

$$e_1 \cap C_j = e_2 \cap C_j \text{ for } j = 0, \ldots, i,$$

where we let $C_0 = \emptyset$. We call the corresponding equivalence classes *(i)-blocks*.

Notice that for $i \geq j$, i-equivalence is a refinement of j-equivalence. Thus, if e_1 and e_2 are in the same (i)-block then they are in the same (j)-block for all $j = 0, 1, \ldots, i-1$. The algorithm proceeds in stages $i = k, k-1, \ldots, 0$, where in stage i the algorithm considers (i)-blocks. More precisely, in stage i the algorithm computes for each pair of (i)-blocks A, A' the coset $ISO(Y, Y')$ of all isomorphisms between the hypergraphs Y and Y' induced by A and A', respectively, on $V_i = C_i \cup \cdots \cup C_k$ and stores this coset in a table T.

Stage k: Let A, A' be two (k)-blocks and let Y, Y' be the corresponding hypergraphs on the vertex set C_k as defined above. Since A and A' are (k)-blocks, the sets $E(Y) = \{e \cap C_k \mid e \in A\}$ and $E(Y') = \{e \cap C_k \mid e \in A'\}$ only contain a single hyperedge a and a', respectively.
Clearly, $ISO(Y, Y') = \emptyset$ if $|a| \neq |a'|$. Otherwise, $ISO(Y, Y') \subseteq \mathrm{Sym}(C_k)$ is the coset of $\mathrm{Aut}(Y) = \mathrm{Sym}(C_k)_a$ that maps a to a' which can be easily computed in time $O(N)$ and stored in the table entry $T[A, A']$.

Stage $i < k$: Let A, A' be two (i)-blocks and let Y, Y' be the corresponding hypergraphs on the vertex set V_i. We explain how to compute the entry $T[A, A'] = ISO(Y, Y')$.
Let a and a' be the unique subsets of C_i such that for all $e \in A$, $e \cap C_i = a$ and for all $e' \in A'$, $e' \cap C_i = a'$. Clearly $ISO(Y, Y')$ is empty if the sizes of a and a' or the sizes of the hyperedge sets $E(Y) = \{e \cap V_i \mid e \in A\}$ and $E(Y') = \{e \cap V_i \mid e \in A'\}$ differ. Otherwise, let $S_1 = \{\varphi \in \mathrm{Sym}(C_i) \mid a^\varphi = a'\}$ be the set containing all permutations in $\mathrm{Sym}(C_i)$ that map a to a' and let S_2 be the set of all permutations on V_{i+1} that map Y to Y' isomorphically *when restricted* to V_{i+1}. Crucially, since A and A' are both (i)-blocks it follows that $ISO(Y, Y') = S_1 \times S_2$.
Clearly, S_1 can be easily computed as explained above. The crux of the algorithm is in computing the set S_2. We first explain a naive method that takes time $(b!)2^{O(b)}N^{O(1)}$ (later we will explain the more complicated $2^{O(b)}N^{O(1)}$ algorithm for computing S_2).

To compute S_2, we partition the (i)-blocks A and A' into $(i+1)$-blocks $A_1, \cdots, A_\ell$ and $A'_1, \cdots, A'_{\ell'}$, respectively. Since S_2 is empty if $\ell \neq \ell'$ we assume $\ell = \ell'$. For each $j = 1, \ldots, \ell$, let Z_j and Z'_j be the hypergraphs induced by the $(i+1)$-blocks A_j and A'_j, respectively, on the vertex set V_{i+1}. Now it is easy to see that

$$S_2 = \bigcup_{\pi \in S_\ell} \bigcap_{j=1}^{\ell} ISO(Z_j, Z'_{\pi(j)}),$$

where the sets $ISO(Z_j, Z'_{\pi(j)})$ are already stored in the table T. Now, observe that instead of cycling through all $\pi \in S_\ell$ it suffices to cycle through all $\rho \in \mathrm{Sym}(C_{i+1})$ and check whether $\{\{e \cap C_{i+1} \mid e \in A\}\}^\rho = \{\{e' \cap C_{i+1} \mid e' \in A'\}\}$. For each such ρ the corresponding induced permutation $\pi \in S_\ell$ with $\{\{e \cap C_{i+1} \mid e \in A_j\}\}^\rho = \{\{e' \cap C_{i+1} \mid e' \in A'_{\pi(j)}\}\}$ can be easily derived.

Now we can apply Theorem 5 to compute for each $\rho \in \mathrm{Sym}(C_{i+1})$ which corresponds to some $\pi \in S_\ell$ as explained above the coset intersection $H_\rho \sigma_\rho = \bigcap_{j=1}^{\ell} ISO(Z_j, Z'_{\pi(j)})$ which is either empty or a coset. As $\ell \leq 2^b$, this takes time bounded by $2^{O(b)}N^{O(1)}$. Now the algorithm can compute

$$S_2 = \bigcup_{\rho \in \mathrm{Sym}(C_{i+1})} H_\rho \sigma_\rho$$

which again is either empty or a coset and stores the set $S_1 \times S_2$ in $T[A, A']$.

Since there is a single (0)-block E, we can find $\mathrm{Aut}(X) = T(E, E)$ in the table. It remains to analyze the running time of the algorithm. The number of blocks at any stage is bounded by the number of edges of X. Thus, the i-th stage takes time bounded by $b!2^{O(b)}N^{O(1)}$, where the $b!$ factor is because we cycle through all the $\rho \in \mathrm{Sym}(C_{i+1})$.

In order to obtain the improved $2^{O(b)}N^{O(1)}$ time bound, it suffices to give a $2^{O(b)}N^{O(1)}$ time algorithm for computing the coset S_2 of all permutations on V_{i+1} that map Y to Y' isomorphically *when restricted* to V_{i+1}.

Claim 8. *There is a $2^{O(b)}N^{O(1)}$ time algorithm for computing S_2.*

We will compute S_2 with a dynamic programming strategy that will involve solving $2^{O(b)}$ many subproblems and $2^{O(b)}$ many coset intersection instances for which we can invoke Theorem 5. We use ideas from Luks's dynamic programming algorithm in [16]. For each subset $\Delta \subseteq C_{i+1}$ and $\Sigma \subseteq C_{i+1} \setminus \Delta$ we define hypergraphs

$$\begin{aligned} Y^{\Delta,\Sigma} &= \{e \cap V_{i+1} \mid e \in Y, e \cap (C_{i+1} \setminus \Delta) = \Sigma\}, \text{ and} \\ Y'^{\Delta',\Sigma'} &= \{e' \cap V_{i+1} \mid e' \in Y', e' \cap (C_{i+1} \setminus \Delta') = \Sigma'\}. \end{aligned}$$

Notice that Y projected on V_{i+1} is $Y^{C_{i+1},\emptyset}$ and that Y' projected on V_{i+1} is $Y'^{C_{i+1},\emptyset}$, and we are interested in computing $S_2 = ISO(Y^{C_{i+1},\emptyset}, Y'^{C_{i+1},\emptyset})$. Furthermore, notice that for different subsets Σ and Σ' the hypergraphs $Y^{\emptyset,\Sigma}$ and $Y^{\emptyset,\Sigma'}$ are the hypergraphs induced by the different $(i+1)$-blocks. Observe that in the $(i+1)^{st}$ stage we have already computed the cosets $ISO(Y^{\emptyset,\Sigma}, Y'^{\emptyset,\Sigma'})$ for different Σ and Σ' (as these correspond to the different $(i+1)$-blocks). Our goal is to compute all the cosets

$$ISO(\Delta, \Sigma, \Delta', \Sigma'),$$

consisting of all isomorphisms π from the hypergraph $Y^{\Delta,\Sigma}$ to the hypergraph $Y'^{\Delta',\Sigma'}$ that map Δ to Δ' and Σ to Σ'. To this end we actually compute for different subsets $\Gamma \subseteq \Delta$ and $\Gamma' \subseteq \Delta'$ the cosets

$$ISO(\Delta, \Sigma, \Gamma, \Delta', \Sigma', \Gamma') = ISO(\Delta, \Sigma, \Delta', \Sigma') \cap \mathrm{Coset}(\Gamma, \Gamma'), \tag{1}$$

where $\mathrm{Coset}(\Gamma, \Gamma')$ denotes the coset of all permutations on V_{i+1} that map Γ to Γ'. Notice that $\mathrm{Coset}(\Gamma, \Gamma')$ can be easily computed in time $O(N)$. To complete the description of the dynamic programming algorithm, we consider different cases for $|\Gamma|$ and $|\Gamma'|$. Clearly, if $|\Gamma| \neq |\Gamma'|$ then the corresponding coset intersection of Equation 1 is the empty set.

Suppose $|\Gamma| = |\Gamma'| = \ell > 1$. In this case, we fix a subset Γ_1 of Γ of size $\lceil \ell/2 \rceil$ and cycle through all possible subsets Γ'_1 of Γ' of size $\lceil \ell/2 \rceil$. Clearly, we can write

$$ISO(\Delta, \Sigma, \Gamma, \Delta', \Sigma'\Gamma') = \bigcup_{\Gamma'_1 \subset \Gamma'} (ISO(\Delta, \Sigma, \Gamma_1, \Delta', \Sigma', \Gamma'_1) \cap ISO(\Delta, \Sigma, \Gamma \setminus \Gamma_1, \Delta', \Sigma', \Gamma' \setminus \Gamma'_1)),$$

where the union runs over subsets of size $\lceil \ell/2 \rceil$. Computing this union as a coset essentially involves solving at most 2^b many coset intersections, each of which takes $2^{O(b)}N^{O(1)}$ time, assuming that the dynamic programming table entries for Γ_1 and Γ'_1 are already there. Finally, we turn to the case when $|\Gamma| = |\Gamma'| = 1$. Let $\Gamma = \{\gamma\}$ and $\Gamma' = \{\gamma'\}$. Let $\Delta_1 = \Delta \setminus \{\gamma\}$ and $\Delta'_1 = \Delta' \setminus \{\gamma'\}$. It is easy to see that

$$ISO(\Delta, \Sigma, \{\gamma\}, \Delta', \Sigma', \{\gamma'\}) = \mathrm{Coset}(\{\gamma\}, \{\gamma'\}) \cap ISO(\Delta_1, \Sigma \cup \{\gamma\}, \Delta_1, \Delta'_1, \Sigma' \cup \{\gamma'\}, \Delta'_1) \cap ISO(\Delta_1, \Sigma, \Delta_1, \Delta'_1, \Sigma', \Delta'_1),$$

which is again a coset intersection instance for table entries already computed since they correspond to smaller size sets Δ_1 and Δ'_1.

To complete the proof (of both the claim and the theorem), notice that we compute the table entries for increasing sizes of Δ. For each Δ we compute the entries for different Σ and increasing sizes of Γ. Finally, the base case for which the cosets in the table are already computed is when Δ is the empty set. For different subsets Σ these correspond to the $(i+1)$-blocks. This proves the correctness and the running time bound follows from the fact that the number of subproblems is $2^{O(b)}N^{O(1)}$, each of which involves $2^{O(b)}N^{O(1)}$ many coset intersections which takes $2^{O(b)}N^{O(1)}$ time by Theorem 5. ◀

It is easy to modify the algorithm in the above theorem to compute all isomorphisms between two colored hypergraphs $X = (V, E)$ and $X' = (V', E')$ without changing the running time. Clearly, we can assume that $V = V' = C_1 \uplus \cdots \uplus C_k$. The new algorithm computes for each pair of (i)-blocks A, A' the set $ISO(Y, Y')$, where Y and Y' are the hypergraphs induced by A and A', respectively, with the only difference that now the block A comes from the hypergraph X and A' comes from X'. Thus, in stage 0 the algorithm computes the set $ISO(X, X')$ of all isomorphisms from X to X'.

▶ **Corollary 9.** *Let $X = (V, E)$ and $X' = (V, E')$ be two colored hypergraphs of size N with $V = C_1 \uplus \cdots \uplus C_k$ where $|C_i| \leq b$ for all i. Given X and X' as input there is an algorithm that computes the set $ISO(X, X')$ of isomorphisms from X to X' in time $2^{O(b)}N^{O(1)}$.*

5 Discussion

We now briefly address the complexity of the canonization problem associated with CHI. We first recall the definition of canonization. Let $\mathcal{K}$ denote the set of all instances of CHI. A mapping $f : \mathcal{K} \to \mathcal{K}$ is a *canonizing function* for $\mathcal{K}$ if for all pairs of isomorphic instances X and X' in $\mathcal{K}$, $f(X) = f(X')$ and $f(X) \cong X$. We say that f assigns a *canonical form* to each isomorphism class of $\mathcal{K}$.

It is often the case that canonization and isomorphism testing for a class of structures have the same complexity. However, for CHI we do not know a canonization procedure even with running time $(b!)2^{O(b)}n^{O(1)}$. Indeed, we do not know if the problem is fixed parameter

tractable. The following result is the best we know which follows easily by applying known techniques [6].

▶ **Theorem 10.** *The canonization problem associated with* CHI *has an* $N^{O(b)}$ *time algorithm, where* N *is the input size and* b *bounds the size of the color classes.*

Proof Sketch. Let $X = (V, E)$ be an input instance of CHI, where $|E| = m$ and $|V| = n$. Then, by definition, the size of X is $N = mn$. Let $V = \bigcup_{i=1}^{k} C_i$ be the partition of the vertex set into color classes C_i, where $|C_i| \leq b$ for each i. Let $X_i = (V_i, E_i)$ denote the multi-hypergraph obtained from X by projecting the hyperedges $e \in E$ to the set $V_i = C_i \cup \ldots \cup C_k$. The canonization algorithm proceeds inductively. Suppose we have computed the canonical labeling coset $G\sigma$ of the multi-hypergraph X_{i+1}. It suffices to give an $m^{O(b)}$ algorithm for canonizing the multi-hypergraph X_i obtained by projecting E on $V_i = C_i \cup V_{i+1}$, given the canonical labeling coset $G\sigma$ for X_{i+1}. Clearly, it suffices to canonize X_i under the action of the coset $\mathrm{Sym}(C_i) \times G\sigma$, where $\mathrm{Sym}(C_i)$ is the group of the (at most $b!$ many) permutations acting on the color class C_i. Applying the standard orbit finding algorithm for permutation groups [17, 20] we can compute the hypergraph X_i' with vertex set V_i and multiset E_i' consisting of all hyperedges $E_i' = \{\{e^\pi \mid e \in E_i \text{ and } \pi \in \mathrm{Sym}(C_i) \times G\sigma\}\}$. Since $G\sigma$ canonizes X_{i+1}, it follows that $|E_i'| \leq 2^b \cdot |E_i|$. Thus, X_i' can be easily computed in time $\mathrm{poly}(2^b, m, n)$. Notice that every permutation $\pi \in \mathrm{Sym}(C_i) \times G\sigma$ maps X_i to a subgraph X_i^π of the hypergraph X_i'. Furthermore, notice that the automorphism group $\mathrm{Aut}(X_i')$ of X_i' is precisely $\mathrm{Sym}(C_i) \times \sigma^{-1}G\sigma$. Define $Y_i = X_i^{(id,\sigma)}$, where id is the identity permutation in $\mathrm{Sym}(C_i)$. Then, Y_i is clearly a subgraph of X_i', and canonizing X_i under the action of the coset $\mathrm{Sym}(C_i) \times G\sigma$ is equivalent to canonizing Y_i under the action of $\mathrm{Aut}(X_i') = \mathrm{Sym}(C_i) \times \sigma^{-1}G\sigma$. Now, we write the multiset E_i' as

$$E_i' = \{(e_1, n_1), (e_2, n_2), \ldots, (e_r, n_r)\},$$

where the edges e_j are the distinct edges (with corresponding multiplicity n_j), lexicographically ordered. Since $\mathrm{Sym}(C_i) \times \sigma^{-1}G\sigma$ is $\mathrm{Aut}(X_i')$, each permutation in $\mathrm{Sym}(C_i) \times \sigma^{-1}G\sigma$ uniquely defines a permutation on the set $\{e_1, e_2, \ldots, e_r\}$. Thus $\mathrm{Sym}(C_i) \times \sigma^{-1}G\sigma$ gives rise to a subgroup H_i contained in $\mathrm{Sym}(\{e_1, \ldots, e_r\})$. Let $E(Y_i) = \{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$. The hypergraph Y_i, as a subgraph of X_i', can be represented by a *colored* binary string $x \in \{0,1\}^r$, whose j^{th} bit $x_j = 1$ iff $e_j \in E(Y_i)$, and x_j is colored by its multiplicity n_j.

The problem of canonizing X_i under $\mathrm{Sym}(C_i) \times \sigma^{-1}G\sigma$ action reduces to canonize the binary string $x \in \{0,1\}^r$ under the action of the group H_i. Since $\mathrm{Sym}(C_i) \times \sigma^{-1}G\sigma$ is a group with *composition width* [6] bounded by b, it follows that H_i also has composition width bounded by b. Hence, by invoking the Babai-Luks canonization procedure [6] we can compute the canonical form for X_i and the canonical labeling coset in $N^{O(b)}$ time. This completes the proof sketch. ■

References

1. V. Arvind and J. Köbler. On Hypergraph and Graph Isomorphism with Bounded Color Classes. In *Proc. 23rd Symposium on Theoretical Aspects of Computer Science*, volume 3884 of *Lecture Notes in Computer Science*, pages 384–395. Springer-Verlag, 2006.
2. L. Babai. Monte Carlo algorithms for graph isomorphism testing. Technical Report 79-10, Dép. Math. et Stat., Univ. de Montréal, 1979.
3. L. Babai. Moderately exponential bounds for graph isomorphism. In *Proc. International Symposium on Fundamentals of Computing Theory 81*, volume 117 of *Lecture Notes in Computer Science*, pages 34–50. Springer-Verlag, 1981.

4 L. Babai and P. Codenotti. Isomorphism of Hypergraphs of Low Rank in Moderately Exponential Time. In *Proc. 39th Ann. IEEE Symposium on the Foundations of Computer Science*, pages 667–676, IEEE Computer Society Press, 2008.

5 L. Babai, W. Kantor, and E. M. Luks. Computational complexity and the classification of finite simple groups. In *Proc. 24th IEEE Symposium on the Foundations of Computer Science*, pages 162–171. IEEE Computer Society Press, 1983.

6 L. Babai and E. M. Luks. Canonical labeling of graphs. In *Proc. 15th ACM Symposium on Theory of Computing*, pages 171–183, 1983.

7 H. Bodlaender. Polynomial algorithm for graph isomorphism and chromatic index on partial k-trees. *Journal of Algorithms*, 11(4):631–643, 1990.

8 R. G. Downey and M. R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

9 R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer-Verlag, 1999.

10 S. Evdokimov and I. Ponomarenko. Isomorphism of colored graphs with slowly increasing multiplicity of Jordan blocks. *Combinatorica*, 19(3):321–333, 1999.

11 J. Flum and M. Grohe. *Parameterized Complexity Theory*. Springer-Verlag, 2006.

12 M. Furst, J. Hopcroft, and E. M. Luks. Polynomial time algorithms for permutation groups. In *Proc. 21st IEEE Symposium on the Foundations of Computer Science*, pages 36–41. IEEE Computer Society Press, 1980.

13 B. Jenner, J. Köbler, P. McKenzie, and J. Torán. Completeness results for graph isomorphism. *Journal of Computer and System Sciences*, 66:549–566, 2003.

14 S. Kratsch and P. Schweitzer. Isomorphism for graphs of bounded feedback vertex set number is fixed-parameter tractable, 2009.

15 E. M. Luks. Isomorphism of bounded valence can be tested in polynomial time. *Journal of Computer and System Sciences*, 25:42–65, 1982.

16 E. M. Luks. Hypergraph isomorphism and structural equivalence of boolean functions. In *Proc. 31st ACM Symposium on Theory of Computing*, pages 652–658. ACM Press, 1999.

17 E. M. Luks. Permutation groups and polynomial time computations. In L. Finkelstein and W. M. Kantor, editors, *Groups and Computation*, volume 11 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 139–175. American Mathematical Society, 1993.

18 G. L. Miller. Isomorphism testing for graphs of bounded genus. In *Proc. 12th ACM Symposium on Theory of Computing*, pages 225–235. ACM Press, 1980.

19 G. L. Miller. Isomorphism of k-contractible graphs. A generalization of bounded valence and bounded genus. *Information and Computation*, 56(1/2):1–20, 1983.

20 Á. Seress. *Permutation Group Algorithms*. Cambridge University Press, 2003.

21 C. C. Sims. Computational methods in the study of permutation groups. In J. Leech, editor, *Computational problems in abstract algebra, Proc. Conf. Oxford, 1967*, pages 169–183. Pergamon Press, 1970.

22 C. C. Sims. Some group theoretic algorithms. In A. Dold and B. Eckmann, editors, *Topics in Algebra*, volume 697 of *Lecture Notes in Mathematics*, 108–124. Springer 1978.

23 S. Toda. Computing automorphism groups of chordal graphs whose simplicial components are of small size. *IEICE Transactions*, 89-D(8):2388–2401, 2006.

24 K. Yamazaki, H. L. Bodlaender, B. de Fluiter, and D. M. Thilikos. Isomorphism for graphs of bounded distance width. *Algorithmica*, 24(2):105–127, 1999.

25 V. N. Zemlyachenko, N. Konienko, and R. I. Tyshkevich. Graph isomorphism problem (Russian). *The Theory of Computation I, Notes Sci. Sem. LOMI 118*, 1982.

Global Escape in Multiparty Sessions*

Sara Capecchi[1], Elena Giachino[2], and Nobuko Yoshida[3]

1 **Dipartimento di Informatica, Università di Torino**
Corso Svizzera 185, Torino, Italy
capecchi@di.unito.it
2 **Focus Reasearch Team, Università di Bologna/INRIA**
Mura Anteo Zamboni 7, Bologna , Italy
giachino@cs.unibo.it
2 **Imperial College London**
South Kensington Campus, London SW7 2AZ, Great Britain
yoshida@doc.ic.ac.uk

Abstract

This paper proposes a global escape mechanism which can handle unexpected or unwanted conditions changing the default execution of distributed communicational flows, preserving compatibility of the multiparty conversations. Our escape is realised by a collection of asynchronous local exceptions which can be thrown at any stage of the communication and to any subsets of participants in a multiparty session. This flexibility enables to model complex exceptions such as criss-crossing global interactions and fault tolerance for distributed cooperating threads. Guided by multiparty session types, our semantics automatically provides an efficient termination algorithm for global escapes with low complexity of exception messages.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.338

1 Introduction

In multiparty distributed conversations, a frequent communication pattern is the one that provides that some unexpected condition may arise forcing the conversation to abort or to take suitable measures for handling the situation, usually by moving to another stage. Such a *global escape* may be not a computational error but rather a controlled, structured interruption requested by some participant. This paper proposes a structured global escape mechanism based on multiparty session types, which can control multiple interruptions efficiently, and guarantee deadlock-freedom without additional overheads. Our main focus is on *interactional exceptions*, which perform not only local management of the interrupted flows but also explicitly coordinate a set of collaborating and communicating peers. Interactional exceptions based on multiparty sessions provide the following contributions (in which relies the novelty of our approach w.r.t. [4, 6]):

- an extension of multiparty sessions [9] to flexible exception handling: we allow asynchronous escape at any desired point of a conversation, including nested exceptions;
- a flexible exceptions representation for modelling both "light" exceptions, representing a control flow mechanism rather than an error (such as time-outs), and "heavy" exceptions such as component or system crashes;
- a compositional model where nested exception contexts are not a refinement of the outer ones but inner isolated contexts involving only a subset of participants who can handle an unexpected situation without affecting the unrelated communications among other cooperating peers;

* This work has been partially supported by MIUR Projects DISCO (Distribution, Interaction, Specification, Composition for Object Systems), EPSRC EP/G015635/1 and EPSRC EP/F003757/1.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 338–351

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

- exception signals modelled as natural linguistic constructs with a corresponding behaviours;
- applications to large scale protocols among multiple peers, automatically offering communication safety and deadlock-freedom. We apply our theory for well-known distributed protocols for exception handling and resolutions (CAs) [13], and prove our method offers the lower complexity w.r.t. a number of message exchanges than the previously studied algorithms.

As in [4] our extension is *consistent* (since despite synchrony and nesting of exceptions, communications in default and exception handling conversations do not mix) and *safe* (since linearity of communications inside sessions and absence of communication mismatch are enforced carrying out fundamental properties of session types). We ensure these properties using: (i) an asynchronous linguistic construct for exceptions signalling; (ii) multi-level queues: the different levels are used to avoid the mix of messages belonging to standard conversations and exception handling ones, which belong to different nesting levels; (iii) a type discipline based on the known technique of defining global types that describe the whole conversation behaviour, and projected end-point types classifying single peers behaviours.

The paper is organised as follows: in Section 2 we describe the syntax of our calculus and present an example. In Sections 3, 4 we present semantics, typing and properties of our extension; Section 5 shows how a known distributed object model can be encoded with our calculus. Finally Section 6 closes the paper. Detailed definitions, proofs and more examples can be found in [3].

2 Multiparty Session Processes with Exceptions

We introduce the syntax of processes using the $\pi-$calculus with multiparty sessions [9]. The syntax is given in Figure 1, where we use P and Q to range over process names, s over private channels, r over indexed private channels (of the form s^{φ}), a over public channels, v over values, e over expressions, x,y,z over variables, X, Y over term variables and l, l_i over labels. We adopt the notation $\tilde{s}$ as a shorthand for $s_1, \ldots, s_n$.

The session connection is performed by a multicast request $\overline{a}[2..n](\tilde{s}).P$ (over the public channel a, specifying the number n of participants invited) and n accept operations $a[\mathsf{p}](\tilde{s}).P$ (over the same channel a). In both cases $\tilde{s}$ are the private channels that will be used in the continuation. A process engaged in a session can perform an output action $r!\langle e\rangle$, sending on r the evaluation of the expression e, an input action $r?(x).P$, receiving a value on r, bound by x in P. More than one labelled behaviour may be offered on the channel r ($r \rhd \{l_i : P_i\}_{i\in I}$), so that it is possible for a partner to select a behaviour by sending on r the corresponding label ($r \lhd l.P$). The try-catch construct $\mathsf{try}(\tilde{r})\{P\}\ \mathsf{catch}\ \{Q\}$ describes a process P (called *default process*) that communicates on the private channels $\tilde{r}$. If some exception is thrown on $\tilde{r}$ before P has ended, the compensation handler Q is going to take over. The construct $\mathsf{throw}(\tilde{r})$ throws the exception on the channels $\tilde{r}$. Try-catch blocks can be nested, but internal blocks must involve a proper subset of the set of argument channels of the outer try block (this is enforced by the type system). The idea behind this condition is that internal try blocks involve smaller sets of participants using a subset of channels. The exception raised inside these blocks can be resolved by the involved peers without affecting the whole system (or in general a wider sets of participants). If something goes wrong during the execution of the handler, that is the exception cannot be solved "internally" anymore, the control can be passed to the handler of the outer try block by throwing an exception on r (see example 2).

As in [9], in order to model TCP-like asynchronous communications (with non-blocking send but message order preservation between a given pair of participants), we use *queues of messages* L. Queues have a bidimensional structure, this is necessary to guarantee communication consistency: when performing an action on a channel s^{φ}, a process is going to write or read at the level φ of the queue associated to s. However in the user written code the level is always zero ($\varphi = 0$), namely the

P,Q	::=	$\overline{a}[2..n](\tilde{s}).P$	Multicast Request	|	if e then P else P	Conditional
	|	$a[\mathtt{p}](\tilde{s}).P$	Accept	|	$P \mid P$	Parallel
	|	$r!\langle\tilde{e}\rangle$	Output	|	$P; P$	Sequencing
	|	$r?(\tilde{x}).P$	Input	|	$\mathbf{0}$	Inaction
	|	$r \lhd l.P$	Select	|	$(\nu n)P$	Hiding
	|	$r \rhd \{l_i : P_i\}_{i\in I}$	Branch	|	def D in P	Recursion
	|	try$(\tilde{r})\{P\}$ catch $\{P\}$	Try-Catch	|	$X\langle\tilde{e}\tilde{s}\rangle$	Process call
	|	throw$(\tilde{r})$	Throw	|	$s : L$	Named queue

v	::=	$a \mid$ true $\mid$ false	Value	D	::=	$\{X_i(\tilde{x}_i\tilde{s}_i) = P_i\}_{i\in I}$	Declaration
e	::=	$v \mid x \mid e$ and $e' \mid$ not $e \ldots$	Expression	L	::=	$l \cdot L \mid \tilde{v} \cdot L \mid \varnothing$	Queue

Figure 1 Syntax

programmer is not responsible of managing the queue levels, which are increased automatically at runtime during the evaluation (for more detail see Section 3). For a consistent semantics we require that a service can never occur in a try-catch block (this is enforced by the type system), otherwise if an exception is captured and the handler is executed, the session inside the default process will disappear while having still some pending communications. We believe this is only an apparent limitation, because all the practical examples we encountered can be easily implemented in our language.

▶ **Example 1.** We present here a three-party use case that models criss-crossing global interaction [5], which can be coded as in Figure 2 where we use different fonts for *variables* and `values`.

The Seller receives an order from a Client, then, inside a try block, he processes the order and then sends back the confirmation. The Client waits for the order confirmation until, at some point, he decides he has waited too long by throwing an exception. The handler of the Seller checks if the confirmation has been sent by the Seller: if it has not (i.e., *conf* = false) then the interaction is aborted by the Seller; otherwise *conf* = true means that the Client has raised the exception before receiving the confirmation from the Seller; in this case execution goes on with $P_{\mathtt{OK}}$ and $P'_{\mathtt{OK}}$ respectively. Thus *conf* = true corresponds to order completion: from this moment on the interaction must proceed even if the Client has decided differently (this is quite standard in business protocol specifications: when a client aborts too late the transaction he is often compelled either to conclude the payment or to pay some penalty fees). The above interaction contains an escape for the Client who has the right to abort the transaction if the Seller is late for delivery. In $P'_{\mathtt{OK}}$ the Client sends the code of his Bank account to the Bank. The Bank checks if there is enough money then, according to the result of the test, sends `OK` or `NEM` (`Not Enough Money`) to the Client. If the answer is `OK`, the Bank sends `OK` also to the Seller who sends the delivery date to the Client. If the answer is `NEM`, the Client and the Bank start to deal for a loan. Now let us concentrate on the Client-Bank deal. The Bank refreshes the offer every n-seconds, where n is the value of *timeout*. The process iterates until an agreement is reached. The time intervals are modelled through a *timer* construct (let h = `timer`(0) in ...). Thus there are two iterations in the process: one related to the deal (def X) and the other used by the Bank to iterate on time intervals (def Y). The Client examines the Bank offer and, if he agrees on it, he throws an exception to exit the iteration (this is implemented as a inner try-catch block involving only s_2: when the throw is raised the Seller is not involved) otherwise he waits for another offer and iterates the negotiation. Dually the Bank sets the timer to 0 at each deal iteration; the internal recursion iterates until the time-out is reached and then the loan offer is updated. The Bank calculates another offer then iterates the outer recursive process sending the new loan to the Client. An interesting scenario is when the time-out and the acceptance of the loan from the Client rise concurrently. It can then happen that the Client has accepted an offer while the Bank was updating his offer after a time-out: the Bank and the Client agreed on different amounts of money. This is resolved by the handlers P_1

$$
\begin{array}{rcl}
\texttt{Seller} & = & \texttt{BS}[1](s_1, s_2).s_1?(order).\mathsf{try}(s_1, s_2)\{ \text{ elaborate order } ; conf = \texttt{true}; s_1!\langle\texttt{true}\rangle; P_{\texttt{OK}}\} \\
 & & \qquad \mathsf{catch}\ \{\mathsf{try}(s_1, s_2)\{\mathsf{if}\ conf\ \mathsf{then}\ P_{\texttt{OK}}\ \mathsf{else}\ \ \mathsf{throw}(s_1, s_2)\}\ \mathsf{catch}\ \{\texttt{abort}\}\} \\
P_{\texttt{OK}} & = & s_1 \triangleright \{\texttt{OK} : s_1!\langle\texttt{date}\rangle, \texttt{NOK} : \mathsf{throw}(s_1, s_2)\} \\
\texttt{Client} & = & \overline{\texttt{BS}}[2](s_1, s_2).s_1!\langle\texttt{order}\rangle; \mathsf{try}(s_1, s_2)\{s_1?(conf).P'_{\texttt{OK}} \mid \mathsf{throw}(s_1, s_2))\} \\
 & & \qquad \mathsf{catch}\ \{\mathsf{try}(s_1, s_2)\{P'_{\texttt{OK}}\}\ \mathsf{catch}\ \{\texttt{abort}\}\} \\
P'_{\texttt{OK}} & = & s_1!\langle\texttt{code}\rangle; s_1 \triangleright \{\texttt{OK} : s_1?(date).\mathbf{0}, \texttt{NEM} : s_1!\langle\texttt{rf}\rangle; s_1?(f). \\
 & & \mathsf{try}(s_2)\{\mathsf{def}X(s_2) = \mathsf{if}\ \texttt{OK}(f)\ \mathsf{then}\ \mathsf{throw}(s_2)\ \mathsf{else}\ \ s_2?(f).X < s_2 > \mathsf{in}X < s_2 > \mathsf{catch}\{P_1\}\} \\
P_1 & = & s_2!\langle\texttt{f}\rangle; s_1 \triangleright \{\texttt{OK} : s_2?(date), \texttt{NOK} : \mathbf{0}\}\} \\
\texttt{Bank} & = & \overline{\texttt{BS}}[3](s_1, s_2).\mathsf{try}(s_2)\{\mathbf{0}\}\ \mathsf{catch}\ \{\mathsf{try}(s_1, s_2)\{P''_{\texttt{OK}}\}\ \mathsf{catch}\ \{\texttt{abort}\}\} \\
P''_{\texttt{OK}} & = & s_1?(code).\ \mathsf{if}\ enoughmoney\ \mathsf{then}\ s_1 \triangleleft \texttt{OK}.s_1 \triangleleft \texttt{OK}. \\
 & & \ \mathsf{else}\ s_1 \triangleleft \texttt{NEM}.s_1?(rf).s_1!\langle\texttt{f}\rangle; \mathsf{try}(s_2)\{\mathsf{def}X(s_2) = \ \mathsf{let}\ h = \texttt{timer}(0)\ \mathsf{in}\ \mathsf{def}Y(s_2) = \\
 & & \mathsf{if}\ h = timeout\ \mathsf{then}\ \mathsf{calculate}\{\texttt{f}\}; s_2!\langle\texttt{f}\rangle; X < s_2 > \ \mathsf{else} \\
 & & Y < s_2 > \ \mathsf{in}Y < s_2 > \mathsf{in}X < s_2 >\} \ \mathsf{catch}\ \{P_2\} \\
P_2 & = & s_2?(f).\mathsf{if}\ \texttt{OK}(f)\ \mathsf{then}\ s_1 \triangleleft \texttt{OK}.s_1 \triangleleft \texttt{OK}.\ \mathsf{else}\ \ s_1 \triangleleft \texttt{NOK}.s_1 \triangleleft \texttt{NOK}.
\end{array}
$$

Figure 2 Client-Seller-Bank code

and P_2: after the exception has been thrown the Bank sends to the Client the latest value of the loan. The Client checks it and decide whether to accept it or not. In the latter case, being out of money, he sends a NOK label to the Seller who aborts the transaction by throwing an exception.

3 Operational Semantics for Multiparty Exceptions

We extend the semantics of multiparty sessions with exception handling. Exceptions can be raised inside try-catch blocks by means of the throw construct. We ensure that conversations are properly carried on by increasing the level of involved channels in case of exception: handlers communicate on a level φ+1 while pending messages (i.e. that are sent by main processes before passing the execution to the handlers) are sent via channels of level φ. Reduction rules are defined in Figure 3. The reduction system uses an *Exception Environment* Σ, which keeps track of the raised exceptions. This will be used in rules [*Thr*], [*RThr*],[*ZThr*].

Reduction rules use evaluation contexts defined by the following grammars:

$$C := [\,] \mid \mathsf{def}\ D\ \mathsf{in}\ C \mid C; P \qquad \mathcal{E} := C \mid \mathcal{E} \mid P \mid (\nu n)\mathcal{E} \mid \mathsf{try}(\tilde{r})\{\mathcal{E}\}\ \mathsf{catch}\ \{Q\}$$

with the usual semantics: if a process P reduces to P', then $\mathcal{E}[P]$ reduces to $\mathcal{E}[P']$.

Rule [*Link*] establishes the connection among n peers on the private channels $\tilde{s}$, to be used for the communications within the session. One queue for each private channel is produced.

Rules [*Send*], [*Sel*], [*Recv*], [*Branch*] are defined as usual, except for the fact that they put and get values/labels from the φth level of the queue. At first the processes read and write at the first level of the queue ($\varphi = 0$). When a process catches an exception, the indexes of all the occurrences of channels involved are increased by one and the exception is propagated to the other peers, which can safely continue to modify the queues at the previous level until they receive the exception. Those messages delivered to the out-of-date level of the queues will be ignored by the peers that have already caught the exception and increased their queue levels.

Rule [*Thr*] applies when the exception is thrown locally. In this rule a $\mathsf{throw}(\tilde{r})$ is added to the environment in order to acknowledge all the try-catch blocks on the same set of channels $\tilde{r}$. This environment update is performed unless some other $\mathsf{throw}(\tilde{r}')$, with $\tilde{r} \subseteq \tilde{r}'$ is already in Σ, because this would mean that an exception may be caught by an embedding block, causing the current try-catch block to disappear. In that case throwing the exception on channels $\tilde{r}$ would be useless and possibly

$$\Sigma \vdash C_1[\overline{a}[2..n](\tilde{s}).P_1] \mid C_2[a[2](\tilde{s}).P_2] \mid ... \mid C_n[a[n](\tilde{s}).P_n] \longrightarrow \Sigma \vdash (\nu\tilde{s})\,(C_1[P_1] \mid C_2[P_2] \mid ... \mid C_n[P_n] \mid s_1 : \varnothing \mid ... \mid s_m : \varnothing) \quad [Link]$$

$$\Sigma \vdash \mathcal{E}[s^{\varphi}!\langle\tilde{e}\rangle] \mid s[\varphi] : L \longrightarrow \Sigma \vdash \mathcal{E} \mid s[\varphi] : (L :: \tilde{v}) \quad (\tilde{e} \downarrow \tilde{v}) \qquad [Send]$$

$$\Sigma \vdash \mathcal{E}[s^{\varphi} \lhd l.P] \mid s[\varphi] : L \longrightarrow \Sigma \vdash \mathcal{E}[P] \mid s[\varphi] : (L :: l) \qquad [Sel]$$

$$\Sigma \vdash \mathcal{E}[s^{\varphi}?(\tilde{x}).P] \mid s[\varphi] : (\tilde{v} :: L) \longrightarrow \Sigma \vdash \mathcal{E}[P\{\tilde{v}/\tilde{x}\}] \mid s[\varphi] : L \qquad [Recv]$$

$$\Sigma \vdash \mathcal{E}[s^{\varphi} \rhd \{l_i : P_i\}_{i\in I}] \mid s[\varphi] : (l_{i_0} :: L) \longrightarrow \Sigma \vdash \mathcal{E}[P_{i_0}] \mid s[\varphi] : L \quad (i_0 \in I) \qquad [Branch]$$

$$\Sigma \vdash \mathsf{try}(\tilde{r})\{C[\mathsf{throw}(\tilde{r})] \mid P\}\ \mathsf{catch}\ \{Q\} \longrightarrow \Sigma \uplus \mathsf{throw}(\tilde{r}) \vdash \mathsf{try}(\tilde{r})\{C \mid P\}\ \mathsf{catch}\ \{Q\} \qquad [Thr]$$

$$\Sigma, \mathsf{throw}(\tilde{r}) \vdash \mathsf{try}(\tilde{r})\{P\}\ \mathsf{catch}\ \{Q\} \longrightarrow \Sigma, \mathsf{throw}(\tilde{r}) \vdash Q\{s^{\varphi+1}/s^{\varphi}\}_{s^{\varphi}\in\tilde{r}} \quad (\mathsf{throw}(\tilde{r}') \in \Sigma \textit{ implies } \mathsf{try}(\tilde{r}')\ldots \notin P,\ \tilde{r}' \subseteq \tilde{r}) \qquad [RThr]$$

$$\Sigma \vdash (\nu\tilde{s})(\textstyle\prod_i \mathcal{E}_i[\mathsf{try}(\tilde{r})\{\mathbf{0}\}\ \mathsf{catch}\ \{Q_i\}])_{i\in 1..n} \longrightarrow \Sigma \vdash (\nu\tilde{s})(\textstyle\prod_i \mathcal{E}_i)_{i\in 1..n} \quad (\mathsf{throw}(\tilde{r}) \notin \Sigma) \qquad [ZThr]$$

Figure 3 Reduction rules

dangerous (leading to some inconsistency on the queue levels).

The operation of environment update is defined as follows:

$$\Sigma \uplus \mathsf{throw}(\tilde{r}) = \begin{cases} \Sigma & \text{if } \mathsf{throw}(\tilde{r}') \in \Sigma, \natural(\tilde{r}) \subseteq \tilde{r}' \\ \Sigma \cup \mathsf{throw}(\natural(\tilde{r})) & \text{otherwise.} \end{cases}$$

where, $\mathsf{throw}(\tilde{r})$ is added to the environment only if there are no other throw on a bigger or equal set of channels. We use $\natural(\tilde{r})$ to level down the indexes of the channels in $\tilde{r}$, as we can see in the following definition: $\natural(s_1^{\varphi_1}, \ldots, s_n^{\varphi_n}) = s_1^{\varphi}, \ldots, s_n^{\varphi}$, where $\varphi = min(\varphi_1, \ldots, \varphi_n)$.
The need of this operation is made clear by Example 2.

Rule [*RThr*] applies when an exception has been thrown and therefore it can be found in Σ. In this case, the try block reduces to the handler, where all the queue levels are updated. Now let us explain the side condition. The try block reduces only if no inner exception can be caught (i.e., if $\mathsf{throw}(\tilde{r}') \in \Sigma$): this would mean that another peer could be executing the internal handler. In order to be consistent with it, the current process must catch the same internal exception before catching the external one. Then, when all the internal exceptions have been caught, even if a $\mathsf{throw}(\tilde{r}')$ does occur in Σ, it can be ignored, and the handler corresponding to the exception on $\tilde{r}$ can take over.

Rule [*ZThr*] deals with the cases in which the default process in a try block has been reduced to $\mathbf{0}$ and no pending exception occurs in Σ. No such completed try-catch block can be reduced to the inaction, until every other peer has completed the corresponding try-block and are ready to continue the execution. The reason is that even if one try-block has terminated, one among its communicating peers could throw an exception and then the handlers have to interact. So we consider the try-catch blocks in each peer at the same level, when every peer has terminated then they all can go on. Since we consider only well-typed processes, and every process is type-checked with respect to the same global type, it is safe to assume that if we have n communicating peers, then there will be n try-catch blocks to be synchronized.

Rules for conditionals and recursive definitions are standard. Moreover, as usual, we consider processes modulo structural congruence. Besides the standard structural rules, we define the following one: $\mathsf{try}(\tilde{r})\{(\nu n)P\}\ \mathsf{catch}\ \{Q\} \equiv (\nu n)\mathsf{try}(\tilde{r})\{P\}\ \mathsf{catch}\ \{Q\}$ if $n \notin \mathrm{fn}(Q)$.

The complete set of rules can be found in [3].

▶ **Example 2.** Let us consider the reduction of the following process:

$$\emptyset \vdash \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_1)\{\mathsf{throw}(s_1) \mid P\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1, s_2)\}\}\ \mathsf{catch}\ \{Q\} \mid \mathsf{try}(s_1, s_2)\{\mathsf{throw}(s_1, s_2) \mid \mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{Q'\}\}\ \mathsf{catch}\ \{Q''\}$$

In the first line, in the inner try-catch block both the default process and the handler contain a throw and the latter' is on (s_1, s_2). In this way we can model a situation in which if the handling of the enclosed exception fails, the outer block is alerted to handle the failure. The default process in the second line contains a throw on (s_1, s_2).
Case 1 Let us suppose that $\mathsf{throw}(s_1)$ is raised first, by applying rule [*Thr*]:

$$\begin{array}{lll}\mathsf{throw}(s_1) & \vdash & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1,s_2) \mid P'\}\}\ \mathsf{catch}\ \{Q\} \mid \\ & & \mathsf{try}(s_1,s_2)\{\mathsf{throw}(s_1,s_2) \mid \mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{Q'\}\}\ \mathsf{catch}\ \{Q''\}\end{array}$$

Then $\mathsf{throw}(s_1, s_2)$ is raised and we apply [*Thr*] again:

$$\begin{array}{lll}\mathsf{throw}(s_1), \mathsf{throw}(s_1,s_2) & \vdash & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1,s_2) \mid P'\}\}\ \mathsf{catch}\ \{Q\} \mid \\ & & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{Q'\}\}\ \mathsf{catch}\ \{Q''\}\end{array}$$

Then, by rule [*RThr*], the only exception that can be thrown is the one corresponding to $\mathsf{throw}(s_1)$, since, because of the side condition, the external try-blocks on (s_1, s_2) cannot reduce:

$$\begin{array}{lll}\mathsf{throw}(s_1), \mathsf{throw}(s_1,s_2) & \vdash & \mathsf{try}(s_1,s_2)\{\mathsf{throw}(s_1^1,s_2) \mid P'\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q\} \mid \\ & & \mathsf{try}(s_1,s_2)\{Q'\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q''\}\end{array}$$

Notice that, because of the queue level updating, the throw that was part of the inner handler is now $\mathsf{throw}(s_1^1, s_2)$. If something goes wrong in handling the inner exception, $\mathsf{throw}(s_1^1, s_2)$ must reduce even if the level of the channel arguments and of the outer try block do not match: this example shows that the mismatch is due to the fact that some of the channels were involved in a failed exception handling. To balance the levels of the channels in the throw we use the operation $\natural$ which flats all levels to the minimum.
Now we apply rule [*Thr*] again. Since $\mathsf{throw}(\natural(s_1^1, s_2)) = \mathsf{throw}(s_1, s_2)$, the environment Σ does not change:

$$\mathsf{throw}(s_1), \mathsf{throw}(s_1,s_2) \vdash \mathsf{try}(s_1,s_2)\{P'\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q\} \mid \mathsf{try}(s_1,s_2)\{Q'\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q''\}$$

Finally, by rule [*RThr*], the try blocks on s_1, s_2 can reduce:

$$\mathsf{throw}(s_1), \mathsf{throw}(s_1,s_2) \vdash Q\{s_1^2, s_2^1/s_1^1, s_2\} \mid Q''\{s_1^2, s_2^1/s_1^1, s_2\}$$

Notice that the channels in Q and Q'' are all at the same level.

Case 2 Now let us suppose that $\mathsf{throw}(s_1, s_2)$ is raised first:

$$\begin{array}{lll}\mathsf{throw}(s_1,s_2) & \vdash & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1,s_2) \mid P'\}\}\ \mathsf{catch}\ \{Q\} \mid \\ & & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{Q'\}\}\ \mathsf{catch}\ \{Q''\}\end{array}$$

We apply rule [*Thr*] again to the inner $\mathsf{throw}(s_1)$. Notice that $\mathsf{throw}(s_1)$ is not added to the environment since an enclosing throw is already present.

$$\begin{array}{lll}\mathsf{throw}(s_1,s_2) & \vdash & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1,s_2) \mid P'\}\}\ \mathsf{catch}\ \{Q\} \mid \\ & & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1)\{P\}\ \mathsf{catch}\ \{Q'\}\}\ \mathsf{catch}\ \{Q''\}\end{array}$$

then rule [*RThr*] is applied twice:

$$\mathsf{throw}(s_1,s_2) \vdash Q\{s_1^1, s_2^1/s_1, s_2\} \mid Q''\{s_1^1, s_2^1/s_1, s_2\}$$

.

4 Typing Structured Global Escapes

This section extends the type system in [9] to exception handling constructs. In particular the goals of the system are:

(i) to check that the enclosed try-catch block is listening on a smaller set of channels. We want to enforce this condition to enable the independence of the components w.r.t. exceptions: if an exception is captured by an inner component, this is not going to affect the enclosing ones.

(ii) to check that no session request or accept occurs inside a try-catch block as we explained in Section 2.

(iii) to check that throws were written by the programmer in the right position, that is a $\mathsf{throw}(\tilde{r})$ must be at the top level of the default process in a try block on the channels $\tilde{r}$.

For lack of space we cannot be self contained w.r.t. the original system so we just describe the new features. The full type system can be found in [3].

Types. In defining the syntax of our types, we distinguish between *global types*, ranged over by G, which describe the whole communication of a multiparty session, and *end-point types*, ranged over by A, which describe the communication from the point of view of a single participant.

The grammar of a global type is as follows:

Partial	γ	::=	$\mathtt{p}_1 \to \mathtt{p}_2 : k\langle \tilde{S} \rangle \mid \mathtt{p}_1 \to \mathtt{p}_2 : k\{l_i : \gamma_i\}_{i\in I} \mid [\tilde{k}, \gamma, \gamma] \mid \gamma;\gamma \mid \gamma \parallel \gamma \mid \mu \mathbf{t}.\gamma \mid \mathbf{t} \mid \epsilon$
Global	G	::=	$\gamma;\mathsf{end}$
Sorts	S	::=	$\mathsf{bool} \mid \ldots \mid \langle G \rangle$

A global type G is an ended partial type γ. The type $\mathtt{p}_1 \to \mathtt{p}_2 : k\langle \tilde{S} \rangle$ says that participant $\mathtt{p}_1$ sends values of sort $\tilde{S}$ to participant $\mathtt{p}_2$ over the channel k (represented as a natural number). The type $\mathtt{p}_1 \to \mathtt{p}_2 : k\{l_i : \gamma_i\}_{i\in I}$ says that participant $\mathtt{p}_1$ sends one of the labels l_i to participant $\mathtt{p}_2$ over the channel k. If the label l_j is sent, the conversation continues as the corresponding γ_j describes. The type $[\tilde{k}, \gamma, \gamma']$ says that the conversation specified by γ is performed, unless some exception involving channels $\tilde{k}$ arises. In this case the conversation γ' takes over. Moreover, types can be composed by sequential and parallel composition, and they can be recursively defined.

The grammar of an end-point type is as follows:

Partial action	α, β	::=	$k!(\tilde{S}) \mid k?(\tilde{S})$	*send and receive*
		$\mid$	$k \oplus \{l_i : \alpha_i\}_{i\in I} \mid k\&\{l_i : \alpha_i\}_{i\in I}$	*selection and branching*
		$\mid$	$[\tilde{k}, \alpha, \alpha]$	*try-catch*
		$\mid$	$\mu \mathbf{t}.\alpha \mid \mathbf{t}$	*recursion and type variable*
		$\mid$	$\epsilon \mid \alpha;\alpha$	*inaction and sequencing*
Action	A, B	::=	$\alpha \mid \alpha;\mathsf{end} \mid \mathsf{end}$	

As in [9], a session type records the identity number of the session channel it uses at each action type, and we use the *located* type $A@\mathtt{p}$ to represent the end-point type A assigned to participant $\mathtt{p}$.
Types $k!(\tilde{S})$ and $k?(\tilde{S})$ represent output and input of values of type $\tilde{S}$ at s_k. Types $k \oplus \{l_i : \alpha_i\}_{i\in I}$ and $k\&\{l_i : \alpha_i\}_{i\in I}$ describe selection and branching: the former selects one of the labels provided by the latter, say l_i at k then they behave as α_i . The remaining types are a local version of the global ones.

Projection. As usual we define a projection that given a global type G and a participant $\mathtt{p}$ returns the end-point corresponding to the local behaviour of $\mathtt{p}$. We write $G \upharpoonright \mathtt{p}$ to denote such a projection.

The projection is defined first over global types and then over partial global types, that is $(\gamma;\mathsf{end}) \upharpoonright \mathtt{p} = (\gamma \upharpoonright \mathtt{p}).\mathsf{end}$.

The exception type is projected in every participant as a local exception type, even if the participant has no activity in the try-catch block (we call these inactive blocks *dummy try-catch*). This to guarantee that the structure of try-catch blocks is the same in each process.

$$([\tilde{k}, \gamma_1, \gamma_2]) \upharpoonright \mathtt{p} = [\tilde{k}, (\gamma_1 \upharpoonright \mathtt{p}), \gamma_2 \upharpoonright \mathtt{p}] \text{ if } \mathsf{ch}(\gamma_i) \subseteq k \text{ and } \left([\tilde{k}', \gamma_1', \gamma_2'] \in \gamma_i \text{ implies } \tilde{k}' \subset \tilde{k}\right)$$

When the side condition does not hold the map is undefined.

Typing Rules. Type assumptions over names and variables are stored into the *Standard environment* Γ that stores type assumptions over names and variables, and is defined by $\Gamma ::= \emptyset \mid \Gamma, u : S \mid \Gamma, X : \tilde{S}\tilde{A}$, and into the *Session environment* Δ that records session types associated to session channels and is defined by $\Delta ::= \emptyset \mid \Delta, \tilde{k} : \{A@\mathtt{p}\}_{\mathtt{p}\in I}$.

$$\frac{}{\Gamma, a : S \vdash a : S}\ \lfloor\textsc{Name}\rfloor \qquad \frac{\Delta \text{ end only}}{\Gamma \vdash \mathbf{0} \triangleright \Delta \mid \tilde{z}}\ \lfloor\textsc{Inact}\rfloor \qquad \frac{\Gamma, a : \langle G\rangle \vdash P \triangleright \Delta \mid \tilde{z}}{\Gamma \vdash (\nu a)P \triangleright \Delta \mid \tilde{z}}\ \lfloor\textsc{NRes}\rfloor$$

$$\frac{\Gamma \vdash a : \langle G\rangle \quad \Gamma \vdash P \triangleright \Delta \cup \{\tilde{s} : (G \upharpoonright 1)@1\} \mid - \qquad |\tilde{s}| = |\mathsf{sid}(G)|}{\Gamma \vdash \overline{a}[2..n](\tilde{s}).P \triangleright \Delta \mid -}\ \lfloor\textsc{MReq}\rfloor$$

$$\frac{\Gamma \vdash a : \langle G\rangle \quad \Gamma \vdash P \triangleright \Delta \cup \{\tilde{s} : (G \upharpoonright \mathtt{p})@\mathtt{p}\} \mid - \qquad |\tilde{s}| = |\mathsf{sid}(G)|}{\Gamma \vdash a[\mathtt{p}](\tilde{s}).P \triangleright \Delta \mid -}\ \lfloor\textsc{MAcc}\rfloor$$

$$\frac{\forall j . \Gamma \vdash e_j : S_j}{\Gamma \vdash s_k^{\varphi}!\langle \tilde{e}\rangle \triangleright \{\tilde{s} : k^{\varphi}!(\tilde{S})@\mathtt{p}\} \mid \tilde{z}}\ \lfloor\textsc{Send}\rfloor \qquad \frac{\Gamma, \tilde{x} : \tilde{S} \vdash P \triangleright \Delta \cup \{\tilde{s} : A@\mathtt{p}\} \mid \tilde{z}}{\Gamma \vdash s_k^{\varphi}?(\tilde{x}).P \triangleright \Delta \cup \{\tilde{s} : k^{\varphi}?(\tilde{S}); A@\mathtt{p}\} \mid \tilde{z}}\ \lfloor\textsc{Rcv}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta \cup \{\tilde{s} : A_j@\mathtt{p}\} \mid \tilde{z} \qquad j \in I}{\Gamma \vdash s_k^{\varphi} \lhd l_j.P \triangleright \Delta \cup \{\tilde{s} : k^{\varphi} \oplus \{l_i : \alpha_i\}_{i\in I}@\mathtt{p}\} \mid \tilde{z}}\ \lfloor\textsc{Sel}\rfloor \qquad \frac{\Gamma \vdash P_i \triangleright \Delta \cup \{\tilde{s} : A_i@\mathtt{p}\} \mid \tilde{z} \qquad \forall i \in I}{\Gamma \vdash s_k^{\varphi} \rhd \{l_i : P_i\}_{i\in I} \triangleright \Delta \cup \{\tilde{s} : k^{\varphi}\&\{l_i : \alpha_i\}_{i\in I}@\mathtt{p}\} \mid \tilde{z}}\ \lfloor\textsc{Branch}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \{\tilde{s} : \alpha@\mathtt{p}\} \mid \tilde{z} \quad \Gamma \vdash Q \triangleright \{\tilde{s} : \beta@\mathtt{p}\} \mid \tilde{z}' \qquad \mathsf{ch}(P) \subseteq \tilde{z} \quad (\tilde{z}' \neq -) \Rightarrow \tilde{z} \subset \tilde{z}'}{\Gamma \vdash \mathsf{try}(\tilde{z})\{P\}\ \mathsf{catch}\ \{Q\} \triangleright \{\tilde{s} : [\tilde{z}, \alpha, \beta]@\mathtt{p}\} \mid \tilde{z}'}\ \lfloor\textsc{Try}\rfloor \qquad \frac{\natural(\tilde{r}) = \natural(\tilde{z})}{\Gamma \vdash \mathsf{throw}(\tilde{r}) \triangleright \emptyset \mid \tilde{z}}\ \lfloor\textsc{Throw}\rfloor$$

$$\frac{\Gamma \vdash P \triangleright \Delta \mid \tilde{z} \quad \Gamma \vdash Q \triangleright \Delta' \mid \tilde{z} \quad \Delta \asymp \Delta'}{\Gamma \vdash P \mid Q \triangleright \Delta \circ \Delta' \mid \tilde{z}}\ \lfloor\textsc{Par}\rfloor \qquad \frac{\Gamma \vdash P \triangleright \Delta \mid \tilde{z} \quad \Gamma \vdash Q \triangleright \Delta' \mid \tilde{z}}{\Gamma \vdash P; Q \triangleright \Delta \cdot \Delta' \mid \tilde{z}}\ \lfloor\textsc{Seq}\rfloor$$

$$\frac{\Gamma \vdash e : \mathsf{bool} \quad \Gamma \vdash P \triangleright \Delta \mid \tilde{z} \quad \Gamma \vdash Q \triangleright \Delta \mid \tilde{z}}{\Gamma \vdash \mathsf{if}\ e\ \mathsf{then}\ P\ \mathsf{else}\ \ Q \triangleright \Delta \mid \tilde{z}}\ \lfloor\textsc{If}\rfloor$$

$$\frac{\Gamma \vdash \tilde{e} : \tilde{S} \qquad \Delta \text{ end only}}{\Gamma, X : \tilde{S}\tilde{A} \vdash X\langle \tilde{e}\tilde{s}_1..\tilde{s}_n\rangle \triangleright \Delta, \tilde{s}_1 : A_1@\mathtt{p}_1, .., \tilde{s}_n : A_n@\mathtt{p}_n \mid \tilde{z}}\ \lfloor\textsc{Var}\rfloor$$

$$\frac{\Gamma, X : \tilde{S}\tilde{A}, \tilde{x} : \tilde{S} \vdash P \triangleright \tilde{s}_1 : A_1@\mathtt{p}_1, .., \tilde{s}_n : A_n@\mathtt{p}_n \mid \tilde{z} \qquad \Gamma, X : \tilde{S}\tilde{A} \vdash Q \triangleright \Delta \mid \tilde{z}}{\Gamma \vdash \mathsf{def}\ X(\tilde{x}\tilde{s}_1..\tilde{s}_n) = P\ \mathsf{in}\ Q \triangleright \Delta \mid \tilde{z}}\ \lfloor\textsc{Def}\rfloor$$

Figure 4 Typing rules

The typing judgement has the form: $\Gamma \vdash P \triangleright \Delta \mid \tilde{z}$, where P is the process to be typed and $\tilde{z}$ refers to the channels on which the enclosing try-catch block is listening for an exception. We need to take track of those channels in order to ensure (i), (ii) and (iii).

The complete set of typing rules is given in Figure 4. All the rules are standard w.r.t. multiparty sessions theory, except for the two rules $\lfloor\textsc{Try}\rfloor$ and $\lfloor\textsc{Throw}\rfloor$.

In rule $\lfloor\textsc{Try}\rfloor$ the default process P and the exception handler Q are both typed with a session environment composed by $\tilde{s}$ channels only, this is to guarantee that no other communications on channels belonging to some other sessions would be interrupted due to the raising of an exception. The channels $\tilde{z}'$ refers to the channels on which an eventual external try in listening for exceptions. Notice that $\tilde{z}'$ may be an empty sequence, that is the try-block that is being type-checked is at the top level. The default process P considers in its typing the sequence $\tilde{z}$ of the current try-block, while

the exception handler consider the sequence $\tilde{z}'$ of the external try-catch block: this is because when executing the exception handler would be directly enclosed in the external try-catch block. Notice that if $\tilde{z}'$ is different from the empty sequence ($\tilde{z}' \neq -$) then $\tilde{z}$ must be a strict subsequence, this is to guarantee that the current try-catch block is listening on a smaller set of channels w.r.t. the enclosing one. The rule also checks that the channels on which P is communicating ($\mathsf{ch}(P)$) are included in $\tilde{z}$: this is to ensure that all the channels involved in some communication in P will be notified of the exception.

In rule $\lfloor$Throw$\rfloor$ we check that a $\mathsf{throw}(\tilde{r})$ has been written at the right position: it must be at the top level of a default process in a try-block on channels $\tilde{r}$. That is the $\tilde{z}$ recorded in the judgement must be the same as $\tilde{r}$ (up to the $\natural$ operation). Note that given closed annotated processes (i.e. bound variables and names are annotated by types), the type checking is decidable (since checking coherent of G is polymonial with respect to the size of G [7]).

Properties. The type discipline ensures, as in previous session types literature [8, 9]:

- the lack of standard type errors in expressions (***Subject Reduction***):
- communication error freedom (***Communication Safety***),
- the interactions of a typable process exactly follow the specification described by its global type (***Session Fidelity***), and
- once a communication has been established, well-typed programs will never stuck at communication points (***Progress***)

▶ Theorem 4.1 (Subject Congruence and Reduction). Suppose $\Gamma \vdash P \triangleright \emptyset$. Then *(1)* $P \equiv P'$ implies $\Gamma \vdash P' \triangleright \emptyset$; and *(2)* $P \to P'$ implies $\Gamma \vdash P' \triangleright \emptyset$.

For the proof, see [3]. From this theorem, ***Session Fidelity Theorem*** (the interaction of a typable process exactly follow the specification described by its global type) is straightforward. We also state the two theorems which are derived following the technique developed in [9]; see [3] for the proofs.

Below we write $P\langle r! \rangle$ (resp. $P\langle r? \rangle$) if P contains an emitting (resp. receiving) active prefix at r up to $\equiv$, and we say that P has a *redex at* r if it has an active prefix at r among its redexes (Section 5 in [9] gives further details for the redexes). The reduction context $\mathcal{E}$ is defined in Section 3.

▶ Theorem 4.2 (Communication Safety). Suppose $\Gamma \vdash P \triangleright_{\tilde{r}} \Delta$ s.t. Δ is coherent and P has a redex at free s^{φ}. Then:

1. (linearity) $P \equiv \mathcal{E}[s[\varphi] : \tilde{h}]$ such that either
   - **a.** $P\langle s^{\varphi}? \rangle$, s^{φ} occurs exactly once in $\mathcal{E}$ and $\tilde{h} \neq \emptyset$; or
   - **b.** $P\langle s^{\varphi}! \rangle$ and s^{φ} occurs exactly once in $\mathcal{E}$; or
   - **c.** $P\langle s^{\varphi}? \rangle$, $P\langle s^{\varphi}! \rangle$, and s^{φ} occurs exactly twice in $\mathcal{E}$.
2. (error-freedom) if $P \equiv \mathcal{E}[R]$ with $R\langle s^{\varphi}? \rangle$ being a redex:
   - **a.** If $R \equiv s^{\varphi}?(\tilde{y}); Q$ then $P \equiv \mathcal{E}'[s[\varphi] : \tilde{v} \cdot \tilde{h}]$ for some $\mathcal{E}'$ and $|\tilde{v}| = |\tilde{y}|$.
   - **b.** If $R \equiv s^{\varphi} \triangleright \{l_i : Q_i\}_{i \in I}$ then $P \equiv \mathcal{E}'[s[\varphi] : l_j \cdot \tilde{h}]$ for some $\mathcal{E}'$ and $j \in I$.

The type discipline ensures also the progress property. A process is simple when each prefixed subterm in it has only a unique session.

▶ Definition 4.3 (simple). A process P is *simple* when it is typable with a type derivation where the session typing in the premise and the conclusion of each prefix rule is restricted to at most a singleton.

In a simple well-linked P, each session is never hindered by other sessions nor by a name prefixing:

▶ Definition 4.4 (well-linked). We say P is *well-linked* when for each $P \to^* Q$, whenever Q has an active prefix whose subject is a (free or bound) shared name, then it is always part of a redex.

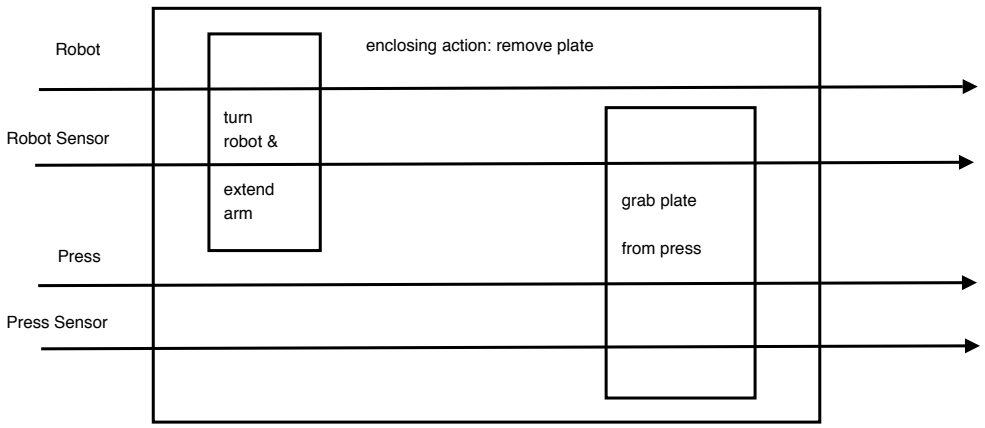


Figure 5 The coordinated action `remove-plate`.

Then the following theorem states that a progress holds for a simple process with a queue for each session channel, such that each prefixed subterm in it has only a unique session and such that each session is never hindered by other sessions nor by a name prefixing.

▶ Theorem 4.5 (Progress). Let P be a simple and well-linked program and $\Gamma \vdash P \triangleright \emptyset$. Then P has the *progress property* in the sense that $P \to^* P'$ implies either $P' \equiv \mathbf{0}$ or $P' \to P''$ for some P''.

5 Coordinated Exception Handling and Resolution: an Example

Coordinated Atomic Actions. In the context of distributed object systems, *Coordinated Atomic Actions* (CA*s*) [13] represent conversation units w.r.t. resource access and recovery activities, in the sense that when an exception is thrown inside a CA the handling is confined within the CA participants, unless something goes wrong during exception handling. Namely two kind of exceptions may be raised: *internal exceptions* $\mathtt{E}$ which can be handled locally and *external exceptions* ϵ that must be signalled to the environment (the enclosing action or the whole system). Disjoint subsets of participants may join nested CAs and, consequently, nested exception contexts. Exceptions can be propagated along chains of nested actions: if the local handling of an exception $\mathtt{E}$ is not successful, then a corresponding exception ϵ will be thrown to the enclosing action. When one or more exceptions are raised in a CA the following actions are performed: (i) the cooperating threads are informed, (ii) nested actions are aborted because $\mathtt{E}$ has been thrown outside them, (iii) during abortion the handler may signal other exceptions, (iv) an algorithm determines which exception must be covered.

CAs have been adapted in [14] to model fault tolerant Web Services: the resulting Web Service Composition Actions (WSCA) relax transactional requirements over external objects since they cannot always be enforced in open systems. In the case of web services these transactional properties can be abstracted and left optional in the various services.

We model the cooperating threads in a CA as a set of participants $\{\mathtt{p}\}_{\{i \in I\}}$ performing a $[Link]$: the established session represents the outermost CA while nested CAs are implemented by try-catch blocks involving only a subset of threads/channels. For each (nested) CA we implement of an "exception resolver" participant. This process is inactive during normal execution, but when one or more exceptions are thrown it collects the corresponding messages from the cooperating threads, decides which exception must be covered, and then sends the corresponding label to all participants. In the following we assume the resolver uses channel 1.

Let $\{\mathrm{E}_h\}_{h\in H}$ and $\{\epsilon_k\}_{k\in K}$ be sets of internal and external exception respectively and

$$(\nu\tilde{s}')\left(\begin{array}{ll} \ldots \mid \mathsf{try}(\tilde{s})\{\mathbf{0}\}\ \mathsf{catch}\ \{Q^1\} & \mid \mathsf{try}(\tilde{s})\{P^2\}\ \mathsf{catch}\ \{Q^2\} \mid \\ \ldots \mid \mathsf{try}(\tilde{s})\{P^n\}\ \mathsf{catch}\ \{Q^n\} & \mid s_1 : \varnothing \mid \ldots \mid s_m : \varnothing \end{array}\right)$$

represent a CA where $\tilde{s} \subset \tilde{s}' = \{s_1, \ldots s_m\}$: $\tilde{s}'$ is the set of channels involved in the enclosing action while $\tilde{s}$ are the channels involved in the nested one (remember that nested actions, i.e. nested try-catch blocks can only involve proper subsets of channels).
The handler of the resolver process is:

$$Q^1 = \mathsf{try}(\tilde{s})\{s_1?(x_2).\ldots.s_1?(x_n).\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q^1_G \quad (1)$$
$$\text{where } Q^1_G = \mathsf{try}(\tilde{s})\{\text{ algorithm determining } h \in H.\ \ s_1!\langle(h)\rangle \ldots s_1!\langle(h)\rangle\}\ \mathsf{catch}\ \{\mathbf{0}\} \quad (2)$$

The handlers Q^j for $j \in \{2, \ldots, n\}$ have the shape:

$$Q^j = \mathsf{try}(\tilde{s})\{\mathsf{if}\ \mathtt{test}\ \mathsf{then}\ s_1!\langle h\rangle\ \mathsf{else}\ \ s_1!\langle 0\rangle\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q^j_G \quad (3)$$
$$\text{and } Q^j_G = \mathsf{try}(\tilde{s})\{s_1?(x).; P^j\}\ \mathsf{catch}\ \{\mathsf{throw}(\tilde{s}')\}, \quad (4)$$

where `test` checks whether exception h has been raised by the current component.

Production Cell. We now implement a part of a case study modelling an industrial Production Cell. This example was proposed as a challenging case study by the FZI in 1993 [12]. The production cell consists of some devices (belts, elevating rotary table, press and rotary robot with two orthogonal extensible arms) associated with a set of sensors, and its task is to get a metal plate from its "environment" via the feed belt, transform it into the forged plate by using a press, and return it to the environment via the deposit belt. For a detailed explanation of the model see [16]. Here we just model the part of the system responsible of removing the plate from the press. The components we are considering are `Robot`, `RobotSensor`, `Press` and `PressSensor` abbreviated respectively as *R, RS, P, PS*. They are cooperating in the `remove-plate` action. There are two nested actions: `turn-robot-and-extend-arm` abbreviated as `TR` and `grab-plate-from-press` abbreviated as `GP`. For the sake of simplicity we assume that the exceptions that can be raised are `Robot-failure`, `Robot-sensor-failure`, `Press-failure`, `Press-Sensor-failure` abbreviated respectively as *RF, RSF, PF, PSF*. The handlers Q are indexed by the participants: Q^{RS} corresponds to the handler associated to the `robot-sensor`. In case of problems during exception handling the control is passed to the enclosing action by signalling one of the following exceptions: `BadRobotRecovery`, `BadRobotSensorRecovery`, `BadPressRecovery` and `BadPressSensorRecovery` abbreviated respectively as *BR, BRS, BP, BPS*.

The enclosing action `remove-plate` uses channels s_1, s_2. Concerning nested actions channel s_1 is used in action `turn-robot-and-extend-arm` while channel s_2 is used in action `grab-plate-from-press`. We recall that we write s as a shorthand for s^0. The action `remove-plate` is then implemented as in the following (where we omit dummy try-catch):

$$\mathtt{ResolverTR} \mid \mathtt{Robot} \mid \mathtt{RobotSensor} \mid \mathtt{Press} \mid \mathtt{PressSensor} \mid \mathtt{ResolverGP} \mid s_1 : L_1 \mid s_2 : L_2 \mid s_3 : L_3$$

where

$$\mathtt{ResolverTR} = \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_1)\{\mathbf{0}\}\ \mathsf{catch}\ \{R^{TR}\}\}\ \mathsf{catch}\ \{\mathbf{0}\}$$
$$\mathtt{ResolverGP} = \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_1)\{\mathbf{0}\}\ \mathsf{catch}\ \{R^{GP}\}\}\ \mathsf{catch}\ \{\mathbf{0}\}$$
$$\mathtt{Robot} = \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_1)\{P^R\}\ \mathsf{catch}\ \{Q^R\}\}\ \mathsf{catch}\ \{Q'^R\}$$
$$\mathtt{RobotSensor} = \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_1)\{P^{RS}\}\ \mathsf{catch}\ \{Q^{RS}\}; \mathsf{try}(s_2)\{P'^{RS}\}\ \mathsf{catch}\ \{Q'^{RS}\}\}\ \mathsf{catch}\ \{Q''^{RS}\}$$
$$\mathtt{Press} = \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_2)\{P^P\}\ \mathsf{catch}\ \{Q^P\}\}\ \mathsf{catch}\ \{Q'^P\}$$
$$\mathtt{PressSensor} = \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_2)\{P^S\}\ \mathsf{catch}\ \{Q^{PS}\}\}\ \mathsf{catch}\ \{Q'^{PS}\}.$$

Let us notice that each nested action has a corresponding resolver. Let us focus on `Robot | RobotSensor` (where we put $P' = \mathsf{try}(s_2)\{P'_{RS}\}\ \mathsf{catch}\ \{Q'^{RS}\}$) and suppose there is a failure in the `Robot` and concurrently in the `Robot-sensor`, that is:

$$\mathtt{ResolverTR} \mid \mathtt{Robot} \mid \mathtt{RobotSensor} \longrightarrow^*$$
$$\mathtt{ResolverTR} \mid \mathsf{try}(s_1, s_2)\{\mathsf{try}(s_1)\{\mathsf{throw}(s_1); P''\}\ \mathsf{catch}\ \{Q^R\}\}\ \mathsf{catch}\ \{Q'^R\} \mid$$
$$\mathsf{try}(s_1, s_2)\{\mathsf{try}(s_1)\{\mathsf{throw}(s_1); P'''\}\ \mathsf{catch}\ \{Q^{RS}\}; P'\}\ \mathsf{catch}\ \{Q'^{RS}\}$$

We apply rule ⌊Thr⌋ and ⌊RThr⌋ twice obtaining:

$$\mathsf{throw}(s_1) \vdash \quad \mathtt{ResolverTR} \mid \mathsf{try}(s_1, s_2)\{Q^R\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q'^R\} \mid$$
$$\mathsf{try}(s_1, s_2)\{Q^{RS}\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q'^{RS}\}$$

After the substitution $\{s_1^1/s_1\}$ we have:

$$\begin{array}{ll}\mathsf{throw}(s_1) \vdash & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{s_1^1?(x_1); s_1^1?(x_2)\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q_G\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{\mathbf{0}\} \mid \\ & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\mathsf{if}\ldots\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q_G^R\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q'^R\} \mid \\ & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\mathsf{if}\ldots\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q_G^{RS}\{s_1^1/s_1\}; P'\}\ \mathsf{catch}\ \{Q'^{RS}\} \mid s_1[1] : (\emptyset),\end{array}$$

we apply both rule ⌊Send⌋ and rule ⌊Rec⌋ twice:

$$\begin{array}{ll}\mathsf{throw}(s_1) \vdash & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\mathbf{0}\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q_G\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{\mathbf{0}\} \mid \\ & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\mathbf{0}\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q_G^R\{s_1^1/s_1\}\}\ \mathsf{catch}\ \{Q'^R\} \mid \\ & \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\mathbf{0}\}\ \mathsf{catch}\ \{\mathbf{0}\}; Q_G^{RS}\{s_1^1/s_1\}; P'\}\ \mathsf{catch}\ \{Q^{RS}\} \mid s_1[1] : (\emptyset).\end{array}$$

Now the execution goes on with the general handlers processes:

$$\begin{array}{l}\mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\text{ algorithm determining } h \in H.\ \ s_1!\langle(h)\rangle\}\ \mathsf{catch}\ \{\mathbf{0}\}\}\ \mathsf{catch}\ \{\mathbf{0}\} \mid \\ \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{s_1?((x));\ \text{informing external objects}\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1^1,s_2)\}\}\ \mathsf{catch}\ \{Q^R\} \mid \\ \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{s_1?((x));\ \text{informing external objects}; P'\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1^1,s_2)\}\}\ \mathsf{catch}\ \{Q^{RS}\} \mid s_1[1] : (\emptyset),\end{array}$$

the resolver reads the value of the received labels, calculates which exception must be covered and sends the corresponding label to the other processes. As explained above the handler of Q_G's processes is a throw on the outer set of channels: the reason is that if an exception is raised during the general handler execution, the exception must be recovered by the enclosing action. Now there are two cases:

1. the execution of the general handlers terminates without problems. In this case the execution goes on with the next nested action `grab-press-from-plate` in which `RobotSensor`, `Press` and `PressSensor` cooperate:

   $$\begin{array}{l}\mathsf{throw}(s_1) \vdash \mathsf{try}(s_1,s_2)\{\mathbf{0}\}\ \mathsf{catch}\ \{Q'^R\} \mid \mathsf{try}(s_1,s_2)\{P'\}\ \mathsf{catch}\ \{Q'^{RS}\} \\ \mathtt{Press} \mid \mathtt{PressSensor} \mid s_1 : \ldots\end{array}$$

2. something goes wrong during the general handlers execution (for instance $Q_G^R \longrightarrow^* \mathsf{throw}(s_1^1)$). Then the corresponding handler alerts the enclosing action by signalling a throw on channels s_1^1, s_2:

   $$\begin{array}{l}\mathsf{throw}(s_1) \vdash \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\mathsf{throw}(s_1^1)\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1^1,s_2)\}\}\ \mathsf{catch}\ \{Q'^R\} \mid \\ \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{Q_G^{RS}\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1^1,s_2)\}; P'\}\ \mathsf{catch}\ \{Q'^{RS}\},\end{array}$$

   we apply rule ⌊Thr⌋:

   $$\begin{array}{l}\mathsf{throw}(s_1), \mathsf{throw}(s_1^1) \vdash \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{\mathbf{0}\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1^1,s_2)\}\}\ \mathsf{catch}\ \{Q'^R\} \mid \\ \mathsf{try}(s_1,s_2)\{\mathsf{try}(s_1^1)\{Q_G^{RS}\}\ \mathsf{catch}\ \{\mathsf{throw}(s_1^1,s_2)\}; P'\}\ \mathsf{catch}\ \{Q'^{RS}\},\end{array}$$

   we apply rule ⌊RThr⌋ twice:

   $$\begin{array}{l}\mathsf{throw}(s_1), \mathsf{throw}(s_1^1) \vdash \mathsf{try}(s_1,s_2)\{\mathsf{throw}(s_1^1,s_2)\}\ \mathsf{catch}\ \{Q'^R\} \mid \\ \mathsf{try}(s_1,s_2)\{\mathsf{throw}(s_1^1,s_2); P'\}\ \mathsf{catch}\ \{Q'^{RS}\}\end{array}$$

   we apply rule ⌊Thr⌋ twice with:

   $$\begin{array}{l}\mathsf{throw}(\natural(s_1^1,s_2)) = \mathsf{throw}(s_1,s_2) \\ \mathsf{throw}(s_1), \mathsf{throw}(s_1^1), \mathsf{throw}(s_1,s_2) \vdash \mathsf{try}(s_1,s_2)\{\mathbf{0}\}\ \mathsf{catch}\ \{Q'^R\} \mid \mathsf{try}(s_1,s_2)\{P'\}\ \mathsf{catch}\ \{Q'^{RS}\},\end{array}$$

   Coming back to the complete action `remove-plate`:

   $$\begin{array}{l}\mathsf{throw}(s_1), \mathsf{throw}(s_1^1), \mathsf{throw}(s_1,s_2) \vdash Q^R\{s_1^2, s_2^1/s_1^1, s_2\} \mid Q^{RS}\{s_1^2, s_2^1/s_1^1, s_2\} \mid \\ Q^P\{s_1^2, s_2^1/s_1^1, s_2\} \mid Q^{PS}\{s_1^2, s_2^1/s_1^1, s_2\}.\end{array}$$

   In this case the execution goes on handling an external exception $i \in \{BR, BRS, BP, BPS\}$. Let us notice that the following nested action, `grab-plate-from-press`, is not executed because of a failure involving the enclosing action.

Correctness and complexity. Let N the number of interacting participants, T_{nmax} be the maximum time of message passing between participants, T_{reso} be the upper bound of the time spent in resolving current exceptions, T_{abort} be the maximum possible time for a thread to abort one nested CA, T_{throw} the cost for signalling the throw (namely to put it in the Σ), $nmax$ be the maximum number of nesting levels of CAs (if no nesting, then $nmax = 0$), Δ_{nmax} be maximum possible time of handling an (resolving) exception. We share with [16] the following results:

1. Any participant p, will complete exception handling ultimately in at most T, where $T = (nmax + 3)T_{nmax} + nmax \cdot T_{abort} + (nmax + 1)(T_{reso} + \Delta_{nmax}) + T_{throw}$.
2. For a given CA A, if no exception is raised in any enclosing action of A, then no more new exceptions will be raised within A once the exception resolution starts.
3. If multiple exceptions are raised concurrently, an ultimate resolving exception that covers all the exceptions will be generated by the proposed algorithm.
4. The number of messages is independent of the number of concurrent exceptions. Taking the nesting of actions into account, in the worst case, our approach requires exactly $nmax(N - 1)$ messages+ N throws. (Let us notice that in [16] the algorithm performs in $O(N^2)$ messages).

6 Conclusions

We have introduced a type-safe global escape mechanism for handling unexpected or unwanted conditions changing the default execution of distributed communicational flows, by means of a collection of asynchronous local exceptions. All the involved conversation parties are guaranteed of the communication safety even after an unforeseen event has been encountered. We have defined a calculus and a type discipline based on the multiparty session [9], and show that the multiparty session types provide a rigorous discipline which can describe and validate complex exception scenarios such as criss-crossing global interactions and fault tolerance for distributed cooperating threads. The flexibility was actually realised allowing local exceptions to be thrown at any stage of the conversation and to any subset of participants. Concerning the criss-cross example our implementation of the protocol never moves to the situation where the Seller sends a confirmation to the Client but the Client aborts the interaction or the Client accepts the wrong loan offer from the Bank.

Related work. Exception handling has been studied for many programming languages including communication-based ones: in distributed object-oriented programming [13, 16], in particular [16] presents the algorithm we implemented as an example in Section 5; several service-oriented calculi (e.g. [2, 11, 15]) include mechanisms for compensation or termination handling, but none of those mechanisms provide a means for coordinating all involved peers that move together to a new stage of the conversation when the unexpected condition is encountered. The paper [10] compares the expressive power of different approaches to compensation; w.r.t. their classification our approach has a static compensation definition, is nested, and has no protection operator. In the context of session types theory, [4, 6] proposed *interactional* exceptions, which inspired our work, for binary sessions and for web service choreographies. The approach described in [4, 6] is significantly different from ours, due to the fact that: (i) exceptions are modelled as special messages exchanged by the parties; (ii) try-catch blocks cannot be at any point in the program but only after a session connection: this means that for a conversation a default behaviour and an exceptional one are defined, while in our calculus try-catch blocks can occur at any point, even nested; and (iii) in those calculi nested try-catch blocks come from nested session connections, and inner exception handlers are refinements of outer ones, while in our case nested try-catch blocks always belong to the same conversation (we forbid session connections inside a try block) and inner exceptions always involve less peers than outer exceptions.

Future work. For the sake of simplicity, so far we have not included in the calculus an important mechanism in the context of session types: session delegation. Even if session delegation seems to be less interesting in the multiparty sessions context than in the binary sessions one, because of the presence of several participants instead of just two, we believe this mechanism is worth of further investigation. Another feature that seems promising w.r.t. practical examples is the capability of distinguishing among different kinds of exceptions (with corresponding different kinds of handlers): the calculus can be easily extended in this direction by putting the right constraints (i.e. all participants

must be able to handle the same set of exceptions). Finally we plan to integrate with multiparty logic work [1] by which we can write a wide range of global escape scenarios which require fine-grained behavioural specifications given by logical assertions, and still ensure communication safety and progress.

Acknowledgments. We thank the FSTTCS reviewers for useful comments. The members of WS-CDL (http://www.w3.org/2002/ws/chor/) and Scribble (http://www.jboss.org/scribble) (in particular, Gary Brown, Kohei Honda and Nickolas Kavantzas) provided many use cases which motivated us to study this subject: we used some of them as the main examples of this paper.

References

1 L. Bocchi, K. Honda, E. Tuosto, and N. Yoshida. A theory of design-by-contract for distributed multiparty interactions. In *CONCUR 2010*, volume 6269 of *LNCS*, pages 162–176. Springer, 2010.

2 L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long-running transactions. In *FMOODS 2003*, volume 2884 of *LNCS*, pages 124–138. Springer, 2003.

3 S. Capecchi, E. Giachino, and N. Yoshida. Global Escape in Multiparty Sessions. Technical report, Imperial College, London, GB, 2010.

4 M. Carbone, K. Honda, and N. Yoshida. Structured interactional exceptions for session types. In *CONCUR 2008*, volume 5201 of *LNCS*, pages 402–417. Springer, 2008.

5 M. Carbone, K. Honda, and N. Yoshida. Theoretical aspects of communication-centred programming. *ENTCS*, 209:125–133, 2008.

6 Marco Carbone. Session-based choreography with exceptions. *ENTCS*, 241:35–55, 2009.

7 P. Deniélou and N. Yoshida. Buffered communication analysis in distributed multiparty sessions. In *CONCUR 2010*, volume 6269 of *LNCS*, pages 343–357. Springer, 2010.

8 K. Honda, V. T. Vasconcelos, and M. Kubo. Language primitives and type disciplines for structured communication-based programming. In *ESOP'98*, volume 1381 of *LNCS*, pages 22–138. Springer, 1998.

9 K. Honda, N. Yoshida, and M. Carbone. Multiparty asynchronous session types. volume 43, pages 273–284, New York, NY, USA, 2008. ACM.

10 I. Lanese, C. Vaz, and C. Ferreira. On the expressive power of primitives for compensation handling. In *ESOP 2010*, volume 6012 of *LNCS*, pages 366–386. Springer, 2010.

11 A. Lapadula, R. Pugliese, and F. Tiezzi. A calculus for orchestration of web services. In *ESOP*, volume 4421 of *LNCS*, pages 33–47. Springer, 2007.

12 C. Lewerentz and T. Lindner, editors. *Formal Development of Reactive Systems - Case Study Production Cell*, London, UK, 1995. Springer-Verlag.

13 C. M. F. Rubira and Zhixue Wu. Fault tolerance in concurrent object-oriented software through coordinated error recovery. In *FTCS '95*, page 499, Washington, DC, USA, 1995. IEEE Computer Society.

14 F. Tartanoglu, V. Issarny, A. Romanovsky, and N. Levy. Coordinated forward error recovery for compositeweb services. *Reliable Distributed Systems, IEEE Symposium on*, 0:167, 2003.

15 H. Torres Vieira, L. Caires, and J. Costa Seco. The conversation calculus: A model of service-oriented computation. In *ESOP 2008*, volume 4960 of *LNCS*, pages 269–283. Springer, 2008.

16 J. Xu, A. Romanovsky, and B. Randell. Coordinated exception handling in distributed object systems: From model to system implementation. In *ICDCS '98*, pages 12–21, Washington, DC, USA, 1998. IEEE Computer Society.

Computationally Sound Abstraction and Verification of Secure Multi-Party Computations *

Michael Backes[1,2], Matteo Maffei[1], and Esfandiar Mohammadi[1]

1 **Saarland University**
{maffei,mohammadi}@cs.uni-saarland.de
2 **Max-Planck Institute for Software Systems**
backes@mpi-sws.org

Abstract

We devise an abstraction of secure multi-party computations in the applied π-calculus. Based on this abstraction, we propose a methodology to mechanically analyze the security of cryptographic protocols employing secure multi-party computations. We exemplify the applicability of our framework by analyzing the SIMAP sugar-beet double auction protocol. We finally study the computational soundness of our abstraction, proving that the analysis of protocols expressed in the applied π-calculus and based on our abstraction provides computational security guarantees.

1998 ACM Subject Classification D.2.4 Software/Program Verification, D.4.6 Security and Protection

Keywords and phrases Computational soundness, Secure multi-party computation, Process calculi, Protocol verification

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.352

1 Introduction

Proofs of security protocols are known to be error-prone and security vulnerabilities have accompanied academic protocols as well as carefully designed and widely deployed products (e.g., Kerberos [10]). Hence work towards the automation of security proofs started soon after the first protocols were developed. From the start, the actual cryptographic operations in such proofs were idealized into so-called Dolev-Yao models [18]. This idealization simplifies proof construction by freeing proofs from cryptographic details such as computational restrictions, probabilistic behavior, and error probabilities. The work on the computational soundness of Dolev-Yao models (e.g., [2, 7, 15, 14, 3])) has largely filled the gap between cryptographic abstractions and computational cryptography, showing that security properties carry over from the former to the latter.

While Dolev-Yao models traditionally comprise only non-interactive cryptographic operations (i.e., cryptographic operations that produce a single message and do not involve any form of communication, such as encryption and digital signatures), recent cryptographic protocols rely on more sophisticated *interactive primitives* (i.e., cryptographic operations that involve several message exchanges among parties), with unique features that go far

* This work was partially funded by the Cluster of Excellence "Multimodel Computing and Interaction" (German Science Foundation), the Emmy Noether Programme (German Science Foundation), the Miur'07 Project SOFT (*Security Oriented Formal Techniques*), the ERC starting grant "End-to-end security", and the DFG grant 3194/1-1.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 352–363

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

beyond the traditional goals of cryptography to solely offer secrecy and authenticity of communication.

Secure multi-party computation (SMPC) constitutes arguably one of the most prominent and most amazing such primitive. Intuitively, in an SMPC, a number of parties $P_1, \ldots, P_n$ wish to securely compute the value $F(d_1, \ldots, d_n)$, for some well-known public function F, where each party P_i holds a private input d_i. This multi-party computation is considered secure if it does not divulge any information about the private inputs to other parties; more precisely, no party can learn more from the participation in the SMPC than she could learn purely from the result of the computation already.

SMPC provides solutions to various real-life problems such as e-voting, private bidding and auctions, secret sharing etc. The recent advent of efficient general-purpose implementations (e.g., FairplayMP [8]) paves the way for the deployment of SMPC into modern cryptographic protocols. Recently, the effectiveness of SMPC as a building block of large-scale and practical applications has been demonstrated by the sugar-beet double auction that took place in Denmark: The underlying cryptographic protocol [9], developed within the Secure Information Management and Processing (SIMAP) project, is based on SMPC.

Given the complexity of SMPC and its role as a building block for larger cryptographic protocols, it is important to develop abstraction techniques to reason about SMPC-based cryptographic protocols and to offer support for the automated verification of their security.

Our contributions. The contribution of this paper is threefold:

- We present an **abstraction** of SMPC within the applied π-calculus [1]. This abstraction consists of a process that receives the inputs from the parties involved in the protocol over private channels, computes the result, and sends it to the parties again over private channels, however augmented with certain details to enable computational soundness results, see below. This abstraction can be used to model and reason about larger cryptographic protocols that employ SMPC as a building block.
- Building upon an existing type-checker [4], we propose an automated **verification technique** for protocols based on our SMPC abstraction. We exemplify the applicability of our framework by analyzing the sugar-beet double auction protocol proposed in [9].
- We establish **computational soundness results** (in the sense of preservation of trace properties) for protocols built upon our abstraction of SMPC. This computational soundness result holds for SMPC that involve arbitrary arithmetic operations; moreover, it is compositional, since the proof is parametric over the other (non-interactive) cryptographic primitives used in the symbolic protocol and within the SMPC itself. Computational soundness holds as long as these primitives are shown to be computationally sound (e.g., in the CoSP framework [3]). We prove in particular the computational soundness of a Dolev-Yao model with public-key encryption, signatures, and the aforementioned arithmetic operations, leveraging and extending prior work in CoSP. Such a result allows for soundly modelling and verifying many applications employing SMPC as a building block, including the case studies considered in this paper.

Related work. Computational soundness was first shown by Abadi and Rogaway in [2] for passive adversaries and symmetric encryption and later extended to active adversaries and additional cryptographic primitives [2, 7, 15, 14, 3]. All these results, however, only consider non-interactive cryptographic primitives, such as encryptions, signatures, and non-interactive zero-knowledge proofs. To the best of our knowledge, our work presents the first computational soundness proof for an interactive primitive.

A salient approach for the abstraction of interactive cryptographic primitives is the Universal Composability framework [11]. The central idea is to define and prove the security of a protocol by comparison with an ideal trusted machine, called the ideal functionality. Although this framework has proven a convenient tool for the modular design and verification of cryptographic protocols, it is not suited to automation of security proofs, given the intricate operational semantics of the UC framework and that ideal functionalities operate on bitstrings (as opposed to symbolic terms). Dolev-Yao models (e.g., the applied π-calculus) offer a higher level of abstraction compared to ideal functionalities in the UC framework. Most importantly, Dolev-Yao models enable automation of security proofs. The different degree of abstraction in these models is best understood by considering digital signatures: While computational soundness proofs for Dolev-Yao abstractions of digital signatures use standard techniques [7, 15, 3] finding a sound ideal functionality for digital signatures has proven to be quite intricate [16]. Yet, securely realizable ideal functionalities constitute a useful tool for proving computational soundness of a Dolev-Yao model. Similarly to [12], we leverage a UC realizability result for showing the computational soundness of our symbolic abstraction.

A generic symbolic abstraction of ideal functionalities has been proposed in [17]. In that work it is shown that the different notions of simulatability, known in the literature, collapse in the symbolic abstraction. In contrast to our approach, that work does not address computational soundness guarantees and does not explicitly consider SMPC.

Outline. Section 2 reviews the applied π-calculus and presents our SMPC abstraction. Section 3 explains the technique used to statically analyze SMPC-based protocols and applies it to our case study. Section 4 presents the computational implementation of a process and studies the computational soundness of our abstraction and Section 5 concludes.

2 The symbolic abstraction of SMPC

In this section, we first review the syntax and the semantics of the calculus. We adopt a variant of the applied π-calculus with destructors [4]. After that, we present the symbolic abstraction of secure multi-party computation within this calculus.

2.1 Review of the applied π-calculus

We briefly review the syntax and the operational semantics of the applied π-calculus, and define the additional notation used in this paper.

Cryptographic messages are represented by *terms*. The set of terms (ranged over by K, L, M, and N) is the free algebra built from names (a, b, c, m, n, and k), variables (x, y, z, v, and w), and function symbols – also called *constructors* – applied to other terms ($f(M_1, \ldots, M_k)$). We let u range over both names and variables. We assume a *signature* Σ, which is a set of function symbols, each with an arity. For instance, the signature may contain the function symbols $enc_{/3}$ of arity 3 and $\mathsf{pk}_{/1}$ of arity 1, representing ciphertexts and public keys, respectively. The term $enc(M, \mathsf{pk}(K), L)$ represents the ciphertext obtained by encrypting M with key $\mathsf{pk}(K)$ and randomness L. We let $\underline{M}$ denote an arbitrary sequence $M_1, \ldots, M_n$ of terms. *Destructors* are partial functions that processes can apply to terms. Applying a destructor d to terms $\underline{M}$ either succeeds and yields a term N (denoted as $d(\underline{M}) = N$), or it fails (denoted as $d(\underline{M}) = \bot$). For instance, we may use the *dec* destructor with $dec(enc(M, \mathsf{pk}(K), L), K) = M$ to model decryption in a public-key encryption scheme.

Plain processes are defined as follows. The null process $\mathbf{0}$ does nothing and is usually omitted from process specifications; $\nu n.P$ generates a fresh name n and then behaves as P;

$a(x).P$ receives a message N from channel a and then behaves as $P\{N/x\}$; $\overline{a}\langle N\rangle.P$ outputs message N on channel a and then behaves as P; $P \mid Q$ executes P and Q in parallel; $!P$ behaves as an unbounded number of copies of P in parallel; $\mathsf{let}\ x = D\ \mathsf{then}\ P\ \mathsf{else}\ Q$ applies if $D = d(\underline{M})$ the destructor d to the terms $\underline{M}$; if application succeeds and produces the term N ($d(\underline{M}) = N$) or D equals a term N, then the process behaves as $P\{N/x\}$; otherwise, i.e., if $d(\underline{M}) = \bot$, the process behaves as Q.[1]

The scope of names and variables is delimited by restrictions, inputs, and lets. We write $fv(P)$ for the free variables and $fn(P)$ for the free names in a process P. A term is *ground* if it does not contain any variables. A process is *closed* if it does not have free variables. A *context* $C[\bullet]$ is a process with a hole $\bullet$. An evaluation context is a context whose hole is not under a replication, a conditional, an input, or an output.

The operational semantics of the applied-π calculus is defined in terms of *structural equivalence* ($\equiv$) and *internal reduction* ($\rightarrow$). Structural equivalence relates the processes that are considered equivalent up to syntactic re-arrangement. Internal reduction defines the semantics of process synchronizations and destructor applications. For more detail on the syntax and semantics of the calculus, we refer to the full version [5].

Safety properties. Following [19], we decorate security-related protocol points with logical predicates and express security requirements in terms of authorization policies. Formally, we introduce two processes $\mathtt{assume}\ F$ and $\mathtt{assert}\ F$, where F is a logical formula. Assumptions and assertions do not have any computational significance and are solely used to express security requirements. Intuitively, a process is safe if and only if all its assertions are entailed by the active assumptions in every protocol execution.

▶ **Definition 1** (Safety). A closed process P is *safe* if and only if for every F and Q such that $P \rightarrow^* \nu\underline{a}.(\mathtt{assert}\ F \mid Q)$, there exists an evaluation context $E[\bullet] = \nu\underline{b}.\bullet \mid Q'$ such that $Q \equiv E[\mathtt{assume}\ F_1 \mid \ldots \mid \mathtt{assume}\ F_n]$, $fn(F) \cap \underline{b} = \emptyset$, and we have that $\{F_1, \ldots, F_n\} \models F$.

A process is *robustly safe* if it is safe when run in parallel with an arbitrary *opponent*.

▶ **Definition 2** (Opponent). A closed process is an *opponent* if it does not contain any assert.

▶ **Definition 3** (Robust Safety). A closed process P is *robustly safe* if and only if $P \mid O$ is safe for every opponent O.

2.2 Abstracting SMPC in the applied π-calculus

We recall that a secure multi-party computation is a protocol among parties $P_1, \ldots, P_n$ to jointly compute the result of a function $\mathcal{F}$ applied to arguments $m_1, \ldots, m_n$, where m_i is a private input provided by party P_i. More generally, not only a function but a reactive, stateful computation is performed, which requires the participants to maintain a (synchronized) state. At the end of the computation, each party should not learn more than the result (or, more generally, a local view r_i of the result). Since the overall protocol may involve several secure multi-party computations, a session identifier *sid* is often used to link the private inputs to the intended session. Coming up with an abstraction of SMPC that is amenable to automated verification and that can be proven computationally sound is technically challenging, and it required us to refine the abstraction based on insights that were gained in the soundness proof. For the sake of exposition, we thus present simple, intuitive abstraction attempts of SMPC

[1] Using a destuctor `equal` that checks term equality, we write $\mathtt{if}\ a = b\ \mathtt{then}\ P\ \mathtt{else}\ Q$ for $\mathtt{let}\ \mathtt{equals}(a, b) = a\ \mathtt{in}\ \mathrm{P}\ \mathtt{else}\ \mathrm{Q}$.

in the applied π-calculus first, explain why these attempts do not allow for a computational soundness result, and then successively refine them until we reach the final abstraction.

First attempt. A first, naive attempt to symbolically abstract SMPC in the applied π-calculus is to let parties send to each other the public information along with the enveloped private input on a private channel. This message can be represented by a term $\mathtt{smpc}(F, i, m, sid)$, where i is the principal's identifier. The abstraction then consists of a destructor `result` whose semantics is defined by a rule like

$$\mathtt{result}(m_i, i, \mathtt{smpc}(\mathcal{F}, 1, m_1, sid), \ldots, \mathtt{smpc}(\mathcal{F}, n, m_n, sid)) = \pi(i, \mathcal{F}(m_1, \ldots, m_n))$$

where $\pi(i, \cdot)$ denotes the projection on the i-th element. This abstraction is unsound: a computational attacker (i) is capable of altering the delivery of messages, and (ii) learns the session identifiers that occur in the header of each individual message. Our abstraction attempt does not grant the adversary any such capabilities; hence a symbolic adversary is much stronger constrained than a computational adversary, thus preventing computational soundness results. These problems could be tackled by modifying the abstraction such that all messages are sent and received over a public channel. The adversary would then decide which parties receive which messages. The resulting abstraction, however, would not be tight enough anymore: Corrupted symbolic parties could send different messages to the participants, and hence cause them to compute the function $\mathcal{F}$ on different inputs. Such an attack, however, is computationally excluded by a secure multi-party computation protocol.

Second attempt. Inspired by the ideal functionality paradigm [11], we solve the aforementioned problems by introducing a trusted party to whom every participant i sends its private input and receives its own result in return over a private channel in_i. A first attempt, called **SMPC_temp**, could look as follows:

$$\begin{aligned}\textbf{SMPC\_temp} \triangleq\ & sidc(sid).in_1(x_1, = sid) \ldots in_n(x_n, = sid). \\ & \mathtt{let}\ (y_1, \ldots, y_n) = \mathtt{result}(\mathcal{F}, x_1, \ldots, x_n)\ \mathtt{in}\ \overline{in_1}\langle y_1, sid\rangle \ldots \overline{in_n}\langle y_n, sid\rangle\end{aligned}$$

There still is discrepancy between this abstraction and the computational model that invalidates computational soundness: a computational adversary learns the session identifier, which is instead concealed by **SMPC_temp**. In addition, the computation of $\mathcal{F}$ is shifted to the evaluation of the destructor `result`. Such a complicated destructor would make mechanized verification extremely difficult.

Final abstraction. To make the abstraction amenable to automated verification, in particular type-checking, we represent $\mathcal{F}$ as a context that explicitly performs the computation. The resulting abstraction of secure multi-party computation is depicted in Figure 1 as the process **SMPC**. This process is parametrized by an adversary channel adv, a session identifier channel $sidc$, n private channels in_i for each of the n participants, and a context $\mathcal{F}$. We implicitly assume that private channels are authenticated such that only the ith participant can send messages on channel in_i. The computational implementation of SMPC implements this authentication requirement. Furthermore, **SMPC** contains two restricted channels for every party i: an internal loop channel $inloop_i$ and an internal input channel lin_i.

SMPC receives a session identifier over the channel $sidc$. Then $n + 1$ subprocesses are spawned: a process $\mathbf{input}_i$ for every participant i that is responsible for collecting the ith input and for divulging public information, such as the session identifier, to the adversary, and a process that performs the actual multi-party computation. Here $\mathbf{input}_i$ waits (under a replication) on the loop channel $inloop_i$ for the trigger message `sync`() of the next round, and

$$
\begin{aligned}
\mathbf{input}_i &\triangleq !\mathit{inloop}_i(z).\mathit{in}_i(x_i, \mathit{sid}').\overline{\mathit{adv}}\langle \mathit{sid}'\rangle.\mathtt{if}\ \ \mathit{sid} = \mathit{sid}' \\
&\quad \mathtt{then}\ \overline{\mathit{lin}_i}\langle x_i\rangle\ \mathtt{else}\ \overline{\mathit{inloop}_i}\langle \mathtt{sync()}\rangle \mid \overline{\mathit{inloop}_i}\langle \mathtt{sync()}\rangle \\
\mathbf{deliver}_i &\triangleq \overline{\mathit{in}_i}\langle y_i, \mathit{sid}\rangle.\overline{\mathit{inloop}_i}\langle \mathtt{sync()}\rangle \\
\mathbf{SMPC}(\mathit{adv}, \mathit{sidc}, \underline{\mathit{in}}, \mathcal{F}) &\triangleq \mathit{sidc}(\mathit{sid}).\nu \underline{\mathit{lin}}.\ \nu \overline{\mathit{inloop}}. \\
&\quad \big(\mathbf{input}_1 \mid \cdots \mid\ \mathbf{input}_n \mid \mathcal{F}[\mathbf{deliver}_1 \mid \cdots \mid \mathbf{deliver}_n]\big)
\end{aligned}
$$

Figure 1 The process **SMPC** as the symbolic abstraction of SMPC

expects the private input x_i and a session identifier sid' over in_i. It then sends the session identifier sid' to the adversary, checks whether the session identifier sid' equals sid, and finally sends the private input x_i on the internal input channel lin_i. The actual multi-party computation is performed in the last subprocess: after the private inputs of the individual parties are collected from the internal input channels lin_i, the actual program $\mathcal{F}$ is executed.

After each computation round, the subprocesses $\mathbf{deliver}_i$ send the individual outputs over the private channels in_i to every participant i along with the session identifier sid. In order to trigger the next round, $\mathtt{sync}()$ is sent over the internal loop channels inloop_i.

The abstraction allows for a large class of functionalities $\mathcal{F}$ as described below.

▶ **Definition 4.** [SMPC-suited context] An *SMPC-suited context* is a context $\mathcal{F}$ such that:

1. $\mathit{fv}(\mathcal{F}) = \{\mathit{sid}\}$ and $\mathit{fn}(\mathcal{F}[\mathbf{0}]) = \{\mathit{lin}_1, \ldots, \mathit{lin}_n\}$.
2. Bound names and variables are distinct and different from free names and free variables.

Although one might expect additional constraints on the context $\mathcal{F}$ (e.g., it is terminating, it does not contain replications, etc.), it turns out that such constraints are not necessary as, intuitively, having more traces in the symbolic setting does not break computational soundness. Such constraints would probably simplify the proof of computational soundness, but they would make our abstraction less intuitive and less general.

Finally, we briefly describe how we model the corruption of the parties involved in the secure multi-party computation. In this paper we consider static corruption scenarios, in which the parties to be corrupted are selected before the computation starts. As usual in the applied π-calculus, we model corrupted parties by letting the adversary know the secret inputs of the corrupted parties. This is achieved by letting the channel in_i occur free in the process if party i is corrupted. As we consider static corruption, we restrict our attention to processes that do not send these channels in_i over a public channel (see Section 4).

Arithmetics in the applied π-calculus. One of the most common applications of SMPC is the evaluation of arithmetic operations on secret inputs. Modelling arithmetic operations in the applied pi-calculus is straightforward. We encode numbers in binary form via the $\mathit{string}_0(M)$, $\mathit{string}_1(M)$, and $\mathit{nil}()$ constructor applications. Arithmetic operations are modelled as destructors. For instance, the greater-equal relation is defined by the destructor $\mathtt{ge}(M_1, M_2)$, which returns M_1 if M_1 is greater equal then M_2, M_2 otherwise. With this encoding, numbers and cryptographic messages are disjoint sets of values, which is crucial for the soundness of our analysis and the computational soundness results.

The Millionaires problem. For the sake of illustration, we show how to express the Millionaires problem in our formalism (two parties wish to determine who is richer, i.e., whose

input is bigger, without divulging their inputs to each other.) The protocol is parametrized by two numbers x_1 and x_2:

$$\begin{aligned} MP &\triangleq \nu sid, sidc, c_1, c_2.\ \overline{adv}\langle sid\rangle.\ \overline{sidc}\langle sid\rangle \mid \mathbf{SMPC}(adv, sidc, \underline{c}, \mathcal{F}) \mid P_1 \mid P_2 \\ P_i &\triangleq \overline{c_i}\langle (x_i, id_i), sid\rangle.c_i(y_i, sid_i) \\ \mathcal{F}[\bullet] &\triangleq lin_1((x_1, z_1)).lin_2((x_2, z_2)).\texttt{let}\ \ z = \mathtt{ge}(x_1, x_2)\ \ \texttt{then}\ \ [y_1 := z, y_2 := z]\bullet \end{aligned}$$

where $[y_1 := z, y_2 := z]$ denotes the instantiation of variables y_1 and y_2 with variable z, which can be defined by encoding, and $\bullet$ is the hole of the context.

3 Formal verification

We propose a technique for formally verifying processes that use the symbolic abstraction **SMPC**. In principle, our abstraction is amenable to several verification techniques for the applied π-calculus such as [19, 4]. In this paper, we rely on the type-checker for security policies presented in [4], which we extend to support arithmetic operations. This type-checker enforces the robust safety property (cf. the full version [5]).

We decorate the protocol linked to our SMPC abstraction with assumptions and assertions. The assumptions are used to mark the inputs of the secure multi-party computation and to specify the correctness property for the secure multi-party computation. The assertion is used to check that the result of the SMPC fulfills the correctness property.

Specifically, for every party i an assumption `assume` $\mathrm{Input}(id_i, x_i, sid)$ is placed upon sending a private input x_i with session identifier sid, where id_i is the (publicly known) identity of party i. To check the correctness of the secure multi-party computation, `assert` $\mathrm{P}(z, sid)$ is placed immediately after the reception of the result (z, sid). We also assume a security policy, which takes in general the following form:

$$\forall \underline{id}, \underline{x}, sid, z.\big(\wedge_{i=1}^{n}\mathrm{Input}(id_i, x_i, sid) \bigwedge_{i,j\in[n]} id_i \neq id_j \wedge F_{rel}\big) \Rightarrow \mathrm{P}(z, sid)$$

The formula F_{rel} characterizes the expected relation between the inputs and the output. As an example, the policy and party annotations for the Millionaires problem would be as follows:

$$\forall id_1, id_2, x, y, sid.\Big(\bigwedge_{i\in\{1,2\}}\mathrm{Input}(id_i, x, sid) \wedge id_1 \neq id_2 \wedge x \geq y\Big) \Rightarrow \mathrm{Richer}(id_1, sid).$$

$$P_i \triangleq \texttt{assume}\ \mathrm{Input}(id_i, x_i, sid) \mid \overline{c_i}\langle (x_i, id_i), sid\rangle.c_i(y_i, sid_i).\texttt{assert}\ \mathrm{Richer}(y_i, sid).$$

Arithmetics in the analysis. We extended the type-checker to support arithmetic operations. Specifically, we modelled arithmetic operations as predicates in the logic, defined their semantics following the semantics given in the calculus, and added a few general properties, such as the transitivity of the greater-equal relation. The type theory is extended to track the arithmetic properties of terms (e.g., while typing $\texttt{let}\ \ z = \mathtt{ge}(x_1, x_2)\ \ \texttt{then}\ \ P$, the type-checker tracks that z is the greatest value between x_1 and x_2 and uses this information to type P). The type theory supports this kind of extensions as long as the set of added values is disjoint from the set of cryptographic messages and the added destructors do not operate on cryptographic terms, which holds true for our encoding of arithmetics.

Case study: sugar-beet double auction. As a case study for our symbolic abstraction, we formalized and analyzed the sugar-beet double auction that has been realized within

the SIMAP project by using an SMPC [9]. This protocol constitutes the first large scale application of an SMPC. The double auction protocol determines a market clearing price for sugar-beets. More specifically, first, a set of prices is fixed; then, for every price each producer commits itself to an amount of sugar-beets that it is willing to sell, and each buyer commits itself to the amounts of sugar-beets that it is willing to buy. The market clearing price is the maximal market clearing price for which the supply did not yet exceed the demand.

Both the producers and the buyers might want to keep their bids secret. Hence, the private input of every party has to be kept private. In the sugar-beet double auction developed by the SIMAP project, the sellers and buyers perform a joint secure multi-party computation. The producer and buyer parties send initially an input and receive at the end of the computation the result, i.e., the market clearing price. In addition, before the start of the protocol, producers and buyers receive a signature on the list of participants, the session identifier, and the set of prices from a trusted party. This signature is verified by each participant and sent as an input to the SMPC, which verifies the signatures and checks whether the data coincides This ensures that all participants agree on the common public inputs. Notice that this SMPC performs arithmetic operations as well as cryptographic operations, yet on distinct values.

Finally, we are ready to state the policy characterizing the result of the computation that is performed: the predicate $\mathrm{MCP}(z, sid)$ holds true if there are appropriate input predicates $\mathrm{Input}(id_i, x_i, sid)$ and z is the maximal price for which demand is greater than or equal to the supply, which we characterize by the predicate $\mathrm{Is\_max}(x_1, \ldots, x_n, z)$ (the semantics of this predicate is defined in terms of basic arithmetic operations supported by our type-checker).

$$\forall z, sid, x_1, \ldots, x_m, id_1, \ldots, id_n.\mathrm{Input}(id_1, x_1, sid) \wedge \cdots \wedge \mathrm{Input}(id_n, x_n, sid) \bigwedge_{i,j \in [n]} id_i \neq id_j \wedge \mathrm{Is\_max}(x_1, \ldots, x_m, z) \Rightarrow \mathrm{MCP}(z, sid)$$

The verification of this policy is challenging in that our abstraction comprises about 1400 lines of code and it relies on complex functions. The type-checker succeeds in 5 minutes and 30 seconds for the SMPC process and the ceritifcation issuer and additional 40 seconds for every participant. [2]

4 Computational soundness of symbolic SMPC

In this section, we establish the computational soundness of our abstraction. We first introduce the computational implementation of a process. Thereafter, we define the notion of robust computational safety. Finally, we state the computational soundness results.

Our computational soundness result is parameterized over the symbolic model $(\mathcal{D}, \mathcal{P})$; this symbolic model consists of a set of constructors and destructors $\mathcal{D}$ (modelling non-interactive primitives such as encryption and decryption) and a class $\mathcal{P}$ of processes.

Computational execution of a process. The semantics of the applied π-calculus is purely symbolic (i.e., it does not involve probabilities, cryptographic messages are represented as symbolic terms instead of bitstrings, the adversary is not computational, etc.). Along the lines of [3], we introduce a probabilistic polynomial-time interactive Turing machine (ITM), called the *computational π-execution*, that interacts with a ppt ITM, called the adversary.

[2] The source code can be found at `http://www.lbs.cs.uni-saarland.de/publications/smpc_simap.spi`. We type checked the protocol with 2 prices, 3 computation parties, and 2 input parties.

The computational π-execution expects as input a process P and a security parameter k. We summarize the main properties of this ITM. It enforces the reduction rules of the operational semantics, draws a random bitstring for each fresh nonce (i.e., restricted name), and executes each constructor and destructor application $d(M)$ (for $d \in \mathcal{D}$) as the computation of the polynomial-time algorithm A_d on input M; A_d is called the implementation of d and $\underline{A}$ is the family of all implementations A_d ($d \in \mathcal{D}$). In the operational semantics of the applied π-calculus, the reduction rules are non-deterministic. This non-determinism is resolved in the implementation by letting the adversary select the reduction steps: The computational execution sends the current process to the adversary and receives back a message indicating the next instruction to be executed. Whenever the execution encounters an `assert` F, it stores a tuple of the form $(F_1, \ldots, F_n, F, \eta, \mu, P)$, called an *assertion tuple*, where F_i are the active assumptions of the current process and η and μ are mappings assigning a bitstring to each variable and name, respectively. We denote the interaction between the computational π-execution using the implementations $\underline{A}$ and an adversary $\mathcal{A}dv$ as $\textsc{Exec}^{\pi}_{P,\underline{A},\mathcal{A}dv}(1^k)$. Given a polynomial p, the distribution of sequences of assertion tuples raised in an interaction of $\textsc{Exec}^{\pi}_{P,\underline{A},\mathcal{A}dv}(1^k)$ within the first $p(k)$ computation steps is denoted as $\mathit{Assertions}^{\pi}_{P,\underline{A},p,\mathcal{A}dv}(k)$.

Our computational π-execution is close in spirit to previously proposed computational executions for the applied π-calculus (e.g., [3]). Such an implementation, however, treats each $\mathbf{SMPC}(\mathit{adv}, \mathit{sidc}, \underline{\mathit{in}}, \mathcal{F})$ abstraction as a trusted host performing a computation and sharing a private channel with each party, whereas the final implementation runs a distributed SMPC protocol. We thus refine the computational π-execution by letting the actual distributed SMPC protocol be executed in place of the (implementation of the) $\mathbf{SMPC}(\mathit{adv}, \mathit{sidc}, \underline{\mathit{in}}, \mathcal{F})$ abstraction. The resulting computational execution $\textsc{Exec}^{\text{SMPC}}_{P,\underline{A},\tau,\mathcal{A}dv}(1^k)$, called *computational SMPC-execution*, is parameterized by the family τ of SMPC protocols implementing each $\mathbf{SMPC}(\mathit{adv}, \mathit{sidc}, \underline{\mathit{in}}, \mathcal{F})$ abstraction. $\mathit{Assertions}^{\text{SMPC}}_{P,\underline{A},\tau,p,\mathcal{A}dv}(k)$ is defined analogous to $\mathit{Assertions}^{\pi}_{P,\underline{A},p,\mathcal{A}dv}(k)$ (see the full version [5] for the formal definitions).

Robust computational safety. The computational notion of robust safety depends on the computational notion of logical entailment. A major difference between the standard symbolic entailment relation and the computational entailment relation is that while the former models destructor application tests $d(\underline{M}) = N$ via logical predicates of the form $\mathrm{Red}(d^{\#}(\underline{M}), N)$ (whose semantics follows from the symbolic rules of the applied π-calculus), the latter computationally checks $A_d(\tilde{\underline{M}}) = \tilde{N}$, $\tilde{\underline{M}}$ and $\tilde{N}$ being the bitstrings computed for $\underline{M}$ and N using η, μ, and $\underline{A}$. Thus, we denote the computational entailment relation as $\models_{\eta,\mu,\underline{A}}$. We now introduce two definitions of robust computational safety, with respect to $\textsc{Exec}^{\pi}$ and $\textsc{Exec}^{\text{SMPC}}$, respectively.

▶ **Definition 5** (Robust computational safety)**.** Let P be a process, $\underline{A}$ an implementation of the destructors in P, and τ a family of secure multi-party computations. We say that P is π-(resp. SMPC-)robustly computationally safe using $\underline{A}$ (resp. $\underline{A}, \tau$) iff for all polynomial-time interactive machines $\mathcal{A}dv$ and all polynomials p, $\Pr[\text{for all } ((F_1, \ldots, F_n), F, \eta, \mu, Q) \in a, \{F_1, \ldots, F_n\} \models_{\eta,\mu,\underline{A}} F : a \leftarrow \mathit{Assertions}^{\pi}_{P,\underline{A},p,\mathcal{A}dv}(k)]$ (resp. $\Pr[\text{for all } ((F_1, \ldots, F_n), F, \eta, \mu, Q) \in a, \{F_1, \ldots, F_n\} \models_{\eta,\mu,\underline{A}} F : a \leftarrow \mathit{Assertions}^{\text{SMPC}}_{P,\underline{A},\tau,p,\mathcal{A}dv}(k)]$) is overwhelming in k.

A symbolic model is computationally sound if robust safety carries over to the computational setting. This definition is used in our first theorem, which is parameterized over the non-interactive primitives used in the protocol.

▶ **Definition 6** (Computationally sound model)**.** Let $\underline{A}$ be a set of constructor and destructor implementations. We say that a symbolic model $(\mathcal{D}, \mathcal{P})$ is computationally sound using $\underline{A}$ iff for all $P \in \mathcal{P}$ such that P is robustly safe, P is π-robustly computationally safe using $\underline{A}$.

There are properties for which computational soundness cannot be shown. For a function H whose range is smaller than its domain (such as a collision-resistant hash function), consider the injectivity property $\forall x, y.x \neq y \Rightarrow H(x) \neq H(y)$. Symbolically, this property holds true if H is a constructor. Computationally, however, this formula naturally does not hold as x and y are universally quantified and collisions cannot be avoided. To exclude such cases, we only consider quantification over protocol messages. This is formalized by requiring that all quantified subformulas $\forall x.F$ and $\exists x.F$ are of the form $\forall x.p(\ldots, x, \ldots) \Rightarrow F'$ and $\exists x.p(\ldots, x, \ldots). \wedge F'$ (where p is a predicate) and we call the resulting class of first-order formulas *well-formed formulas* (see the technical report [5] for a formal definition). Hence $\forall x, y.x \neq y \Rightarrow H(x) \neq H(y)$ is not well-formed. Instead, $\forall x, y.p(x) \wedge p(y) \wedge x \neq y \Rightarrow H(x) \neq H(y)$[3] (where p is a predicate, meant to be assumed upon reception of x and y) is well-formed. As we exclude assumptions of the form $\forall x.p(x)$, $p(m)$ has to be assumed explicitly and m must be a protocol message. For a collision-resistant hash function H, the above formula holds computationally. We stress that in contrast to other computational soundness results (e.g., [3]), where formulas are assumed to be term-free (i.e., contain nullary predicates only), our result holds for formulas containing terms of the calculus and destructor application tests.

In addition we require that (i) restricted channels in_i are never output; (ii) session identifiers are sent at most once on session identifier channels; and (iii) for all subprocesses $\mathbf{SMPC}(adv, sidc, \underline{in}, \mathcal{F})$ of P, the context $\mathcal{F}$ is SMPC-suited. Processes fulfilling this constraints are called *well-formed processes*. A *well-formed class of processes* only contains well-formed processes and enjoys some closeness properties, such as closeness under subprocesses or top-level name restrictions (we refer to the full version [5] for more details).

Computational soundness of symbolic SMPC. We now state the main computational soundness result of this work: the robust safety of a process using non-interactive primitives and our SMPC abstraction carries over to the computational setting, as long as the non-interactive primitives are computationally sound. This result ensures that the verification technique from Section 3 provides computational safety guarantees. We stress that the non-interactive primitives can be used both within the SMPC abstractions and within the surrounding protocol. The proof uses a result from [13] on generic UC-realizable MPC protocols that only holds under standard cryptographic assumptions: the setup assumption of the *CRS-model*[4] and the existence of *enhanced trapdoor permutations*. Moreover, as also required in [3], all implementations $\underline{A}$ have to be *length-regular*.

▶ **Theorem 7** (Computational soundness of symbolic SMPC). *Let $(\mathcal{D}, \mathcal{P})$ be computationally sound, well-formed model using $\underline{A}$, where $\underline{A}$ is length-regular. If enhanced trapdoor permutations exist, then there is a family ρ of SMPC implementations in the CRS-model such that for each well-formed $P \in \mathcal{P}$, the robust safety of P implies the SMPC-robust computational safety of P using $\underline{A}, \rho$.*

Computational soundness of arithmetic operations. We show that our computational soundness result applies to a large class of protocols by proving the computational soundness of a symbolic model with public-key encryption, signatures, and arithmetics[5]. We recall that

[3] This formula is logically equivalent to $\forall x.p(x) \Rightarrow \forall y.p(y) \Rightarrow x \neq y \Rightarrow H(x) \neq H(y)$. For the sake of a clear presentation, the policies in Section 3 are not well-formed, but adding for every quantified variable x a predicate $p(x)$ to the premise makes these policies well-formed as well.

[4] More precisely, in each SMPC session all parties have access to a shared bitstring, called the CRS.

[5] This result extends prior work [3] in the CoSP framework to arithmetics and first-order logic formulas.

numbers are modelled as symbolic bitstrings and arithmetic operations as destructors on these bitstrings. In this way, we enforce a strict separation between numbers and cryptographic terms, such as signatures and keys. Thus, it is not possible to apply an arithmetic operation on nonces, signatures, or encrypted messages. Due to this separation, the impossibility result for computational soundness of XOR [6] does not apply. As in [3], we impose some standard conditions on protocols to ensure that all encryptions and signatures are produced using fresh randomness and that secret keys are not sent around. A protocol satisfying these conditions is called *key-safe*. We denote the resulting symbolic model as $(\mathcal{D}_{\mathrm{ESA}}, \mathcal{P}_{\mathrm{ESA}})$.

The computational soundness proof for $(\mathcal{D}_{\mathrm{ESA}}, \mathcal{P}_{\mathrm{ESA}})$ follows almost exactly the lines of the symbolic model for *key-safe* protocols in the work of Backes, Hofheinz, and Unruh [3]. The proof can be found in the full version [5].

▶ **Theorem 8** (Computational soundness of $(\mathcal{D}_{\mathsf{ESA}}, \mathcal{P}_{\mathsf{ESA}})$)**.** *If enhanced trapdoor permutations exist, there is an length-regular implementation $\underline{A}$ such that $(\mathcal{D}_{ESA}, \mathcal{P}_{ESA})$ is a computationally sound, well-formed model.*

As a corollary of Section 7 and Theorem 8, we get the computational soundness of protocols using SMPC, public-key encryption, signatures, and arithmetics, with such non-interactive cryptographic primitives possibly used within both SMPC and the surrounding protocol.

▶ **Corollary 9** (SMPC computational soundness with $(\mathcal{D}_{\mathsf{ESA}}, \mathcal{P}_{\mathsf{ESA}})$)**.** *There is an implementation $\underline{A}$ and a family of UC-protocols τ such that for each well-formed $P \in \mathcal{P}_{ESA}$, the robust safety of P implies the SMPC-robust computational safety of P using $\underline{A}$ and τ.*

5 Conclusion

We have presented an abstraction of SMPC in the applied π-calculus. We have shown that the security of protocols based on this abstraction can be automatically verified using a type system, and we have established computational soundness results for this abstraction including SMPC that involve arbirtrary arithmetic operations. This is the first work to tackle the abstraction, verification, and computational soundness of protocols based on an interactive cryptographic primitive.

Our framework allows for the verification of protocols incorporating SMPC as a building block. In particular, the type-checker ensures that the inputs provided by the participants to the SMPC are well-formed and, after verifying the correctness of SMPC, the type-checker obtains a characterization of the outputs (e.g., in the Millionaires problem, the output is the identity of the participant providing the greatest input) that can be used to establish global properties of the overall protocol. Our framework is general and covers SMPC based on arithmetic operations as well as on cryptographic primitives.

This work focuses on trace properties, which include authenticity, integrity, authorization policies, and weak secrecy (i.e., the attacker cannot compute a certain value)[6]. As a future work, we plan to extend our framework to observational equivalence relations, possibly building on recent results for a fragment of the applied π-calculus [14]. It would be interesting to formulate and verify an indistinguishability-based secrecy property for SMPC. This task is particularly challenging since the result of an SMPC typically reveals some information about the secret inputs and standard secrecy definitions based on observational equivalence are

[6] To symbolically model weak secrecy, one can add the process c(x).if x=n then assert false (where c is a public channel) to check the secrecy of n: The protocol is robustly safe only if the attacker does not learn n.

thus too restrictive. An additional problem is the computational soundness of observational equivalence for processes with private channels, which are excluded in [14] but are needed in our abstraction and, arguably, in any reasonable SMPC abstraction.

References

1 M. Abadi and C. Fournet. Mobile values, new names, and secure communication. In *POPL*, pages 104–115. ACM Press, 2001.

2 Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). *Journal of Cryptology*, 15(2):103–127, 2002.

3 Michael Backes, Dennis Hofheinz, and Dominique Unruh. CoSP: A general framework for computational soundness proofs. In *CCS*, pages 66–78. ACM Press, 2009.

4 Michael Backes, Catalin Hritcu, and Matteo Maffei. Type-checking zero-knowledge. In *CCS*, pages 357–370, 2008.

5 Michael Backes, Matteo Maffei, and Esfandiar Mohammadi. Computationally sound abstraction and verification of secure multi-party computations. Available at: `http://www.infesc.cs.uni-saarland.de/~mohammadi/publications/smpc.pdf`.

6 Michael Backes and Birgit Pfitzmann. Limits of the cryptographic realization of Dolev-Yao-style XOR. In *ESORICS*, volume 3679 of *LNCS*, pages 178–196. Springer, 2005.

7 Michael Backes, Birgit Pfitzmann, and Michael Waidner. A composable cryptographic library with nested operations (extended abstract). In *CCS*, pages 220–230, 2003.

8 Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: a system for secure multi-party computation. In *CCS*, pages 257–266. ACM, 2008.

9 Peter Bogetoft, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas P. Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, Jesper Buus Nielsen, Kurt Nielsen, Jakob Pagter, Michael I. Schwartzbach, and Tomas Toft. Secure multiparty computation goes live. In *Financial Cryptography*, volume 5628 of *LNCS*, pages 325–343. Springer, 2009.

10 F. Butler, I. Cervesato, A. D. Jaggard, A. Scedrov, and C. Walstad. Formal analysis of Kerberos 5. *Theoretical Computer Science*, 367(1):57–87, 2006.

11 Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

12 Ran Canetti and Jonathan Herzog. Universally composable symbolic analysis of mutual authentication and key exchange protocols. In *TCC*, volume 3876 of *LNCS*, pages 380–403. Springer, 2006.

13 Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *STOC*, pages 494–503. ACM, 2002.

14 Hubert Comon-Lundh and Véronique Cortier. Computational soundness of observational equivalence. In *CCS*, pages 109–118, 2008.

15 Véronique Cortier and Bogdan Warinschi. Computationally sound, automated proofs for security protocols. In *ESOP*, pages 157–171, 2005.

16 Anupam Datta, Ante Derek, John Mitchell, Ajith Ramanathan, and Andre Scedrov. Games and the impossibility of realizable ideal functionality. In *TCC*. Springer, 2006.

17 Stéphanie Delaune, Steve Kremer, and Olivier Pereira. Simulation based security in the applied pi calculus. In *FSTTCS*, pages 169–180. Schloss Dagstuhl, 2009.

18 Danny Dolev and Andrew C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

19 C. Fournet, A. D. Gordon, and S. Maffeis. A type discipline for authorization in distributed systems. In *CSF*, pages 31–45. IEEE, 2007.

Model Checking Concurrent Programs with Nondeterminism and Randomization

Rohit Chadha[1], A. Prasad Sistla[2], and Mahesh Viswanathan[3]

1 **INRIA & LSV, ENS Cachan and CNRS, FRANCE**
2 **University of Illinois, Chicago, USA**
3 **University of Illinois, Urbana-Champaign, USA**

Abstract

For concurrent probabilistic programs having process-level nondeterminism, it is often necessary to restrict the class of schedulers that resolve nondeterminism to obtain sound and precise model checking algorithms. In this paper, we introduce two classes of schedulers called *view consistent* and *locally Markovian* schedulers and consider the model checking problem of concurrent, probabilistic programs under these alternate semantics. Specifically, given a Büchi automaton Spec, a threshold $x \in [0, 1]$, and a concurrent program $\mathbb{P}$, the model checking problem asks if the measure of computations of $\mathbb{P}$ that satisfy Spec is at least x, under all view consistent (or locally Markovian) schedulers. We give precise complexity results for the model checking problem (for different classes of Büchi automata specifications) and contrast it with the complexity under the standard semantics that considers all schedulers.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.364

1 Introduction

The use of randomization in concurrent or distributed systems is often key to achieving certain objectives — it is used in distributed algorithms to break symmetry [22] and in cryptographic protocols to achieve semantic security [19]. The formal analysis of such systems has often modelled them as *Markov Decision Processes* [24], that has both nondeterministic and probabilistic transitions.

In Markov Decision Processes (MDPs), the probability of events depends on the way the nondeterministic choices are resolved during a computation. It is customary to resolve the nondeterminism by a *scheduler* or *adversary*, who chooses a probabilistic transition from a state based on the past sequence of states visited during the computation. When verifying MDPs, one considers the worst possible scenario — one checks that no matter which scheduler is chosen, the probabilistic properties of the system hold. Model checking algorithms based on such semantics for MDPs [5, 24] are known, and tools based on these algorithms have been developed that have been used to analyze many examples [1].

Recently, many researchers have observed [13, 12, 6, 15, 11] that in a number of applications, taking such a pessimistic view and considering all possible schedulers, can yield incorrect verification results. The problem arises when one considers a concurrent system where individual processes exhibit both probabilistic and nondeterministic behavior. For such systems, there are certain *perfect information* schedulers that will resolve local process-level nondeterminism based on information that would not be available to the local process, and there by exhibit behavior that is unreasonable. For example, consider the example presented in [18] of two processes "Toss" and "Guess" that do not communicate with each other. The process Toss tosses a fair coin, and Guess guesses (nondeterministically) what the outcome of Toss's coin toss was. Clearly, since Toss and Guess do not communicate, the probability that Guess makes the right guess should be bounded by $\frac{1}{2}$. However under a scheduler that

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 364–375

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

resolves Guess's nondeterminism based on the result of Toss's coin toss, the probability of a correct guess can be as high as 1! (Additional examples can be found in [9].) Therefore, in analyzing concurrent programs, in many cases, it is necessary to restrict attention to certain "reasonable" schedulers that resolve local nondeterminism based only on information that is locally visible to the process.

We call such schedulers to be *view consistent*. More precisely, a view consistent scheduler is the composition of two schedulers — a *global* scheduler that picks the process to schedule, and a *local* scheduler that chooses a probabilistic transition of the process. We assume that the global scheduler can choose the process based on the entire computation thus far. However, the local scheduler's decision must only be based on the local view of the computation. In other words, if σ and τ are computations such that the states as observable to process $\mathcal{P}$ are identical at every step, then the local scheduler for $\mathcal{P}$ must pick the same transition after both σ and τ. Observe that if the individual processes are purely probabilistic, then every scheduler is view consistent; the difference arises only when there is local nondeterminism. A similar class of schedulers called *distributed schedulers* has been considered in [18, 16, 17]. However, there is a subtle difference between distributed schedulers and view consistent schedulers (see Related Work) and the results presented here do not follow from those in [18, 16, 17]. In this paper, we also consider another class of restricted schedulers that we call *locally Markovian*. Locally Markovian schedulers are view consistent schedulers with the additional restriction that the local scheduler's decision only depends on the length of the computation and the current local state, and not on the entire local view of the computation; note, that in a locally Markovian scheduler, the global scheduler can still choose the process to execute based on the entire history. Locally Markovian schedulers are the natural analog in the concurrent case of Markovian schedulers that have been considered in other contexts [23, 4].

In this paper, we investigate the complexity of the verification problem for concurrent programs. We assume that the correctness specification is given by a Büchi automaton Spec, whose input alphabet consists of the states of the program $\mathbb{P}$, and a threshold $x \in [0,1]$. We say $\mathbb{P} \models^{vc}_{\triangleright x} \mathsf{Spec}$ ($\mathbb{P} \models^{lm}_{\triangleright x} \mathsf{Spec}$), where $\triangleright \in \{>, \geq\}$, if under all view consistent schedulers (locally Markovian schedulers) the measure of computations of $\mathbb{P}$ accepted by Spec is $\triangleright x$. Our results are summarized in Figure 1.

We show that the verification problem is in general undecidable, when we restrict to either view consistent or locally Markovian schedulers. When the threshold is 0, both the problems of checking $\mathbb{P} \models^{vc}_{>0} \mathsf{Spec}$ and $\mathbb{P} \models^{lm}_{>0} \mathsf{Spec}$, remain undecidable even when Spec is a deterministic Büchi automaton. For the case when $x \in (0,1)$, the problems of checking $\mathbb{P} \models^{vc}_{>x} \mathsf{Spec}$, $\mathbb{P} \models^{vc}_{\geq x} \mathsf{Spec}$, $\mathbb{P} \models^{lm}_{>x} \mathsf{Spec}$, and $\mathbb{P} \models^{lm}_{\geq x} \mathsf{Spec}$, remain undecidable even when Spec is a *safety* automaton.[1]

We then investigate the complexity of the verification problems left open by the above undecidability results. Namely, we consider the problems of checking Spec that are deterministic or safety automata, when $x = 1$, and of checking safety properties when $x = 0$. We show that many of these problems are indeed decidable, and we characterize their computational complexity precisely. Specifically, we show that the problems of checking $\mathbb{P} \models^{vc}_{=1} \mathsf{Spec}$ and $\mathbb{P} \models^{lm}_{=1} \mathsf{Spec}$ are **PSPACE**-complete, where Spec is a safety automaton; checking $\mathbb{P} \models^{lm}_{=1} \mathsf{Spec}$, when Spec is deterministic, is **EXPSPACE**-complete; and checking $\mathbb{P} \models^{lm}_{>0} \mathsf{Spec}$, when Spec is a safety automaton, is also **EXPSPACE**-complete. The decidability/complexity of check-

[1] A safety automaton is a deterministic Büchi automaton such that all states are accepting except for a unique rejecting state; all transitions from the rejecting state stay in the rejecting state. Every regular safety property can be recognized by such an automaton, and hence the name.

ing $\mathbb{P} \models^{vc}_{=1} \mathsf{Spec}$ when Spec is deterministic, and checking $\mathbb{P} \models^{vc}_{>0} \mathsf{Spec}$ when Spec is a safety property, remain open. However, we show that these model checking problems for view consistent schedulers are **2-EXPTIME**-complete, for two special classes of programs. The first class is that of programs $\mathbb{P}$ where all processes, except possibly one, are purely probabilistic (i.e., have no local nondeterminism). The second class of programs are those where each process has a set of global variables, and some local variables that are private to the process. In addition, we restrict the program to be *mutually exclusive*, that requires that in each global state exactly one process is enabled and we require the specification to be on the *shared state*, that requires that the state of the specification depends only on the history of global states visited.

We contrast the above complexity results with the complexity of the same verification question when we consider *all* schedulers (not just view consistent or locally Markovian schedulers). As previously observed [13], the complexity of verification with respect to perfect information schedulers, is easier. We show that for safety specifications Spec and $x = 1$, the verification problem is **PSPACE**-complete, just like in the case of view consistent and locally Markovian schedulers. All the other verification problems, on the other hand, are **EXPTIME**-complete. Note that, in contrast to the complexity results reported in [5] for MDPs, the blowup in complexity when considering concurrent programs can be explained by the state space explosion problem.

We conclude this introduction by comparing our model of concurrent programs under view consistent schedulers to other probabilistic models for which model checking results are known. Probabilistic automata on infinite strings [3], are a special case of programs under locally Markovian schedulers (see Theorem 3.1 and Lemma 3.6). Partially Observable MDPs [13] are a special case of programs where *all, except possibly one*, processes are purely probabilistic (see discussion in Related Work). Finally, MDPs are equivalent to programs where *all* processes are purely probabilistic. Thus, many of the commonly studied models are special kinds of concurrent programs, and we exploit these connections to prove some upper bounds using translations and embeddings in to these models. Our proofs of lower bounds on the complexities are quite nontrivial and do not follow from any relationships to the above models since our programs are given in a different notation.

The paper is organized as follows. Section 2 contains preliminaries, and definitions of programs and schedulers. Section 3 contains the technical results and the conclusions are presented in Section 4. Motivating examples and proofs of most of the theorems are given in [9].

Related Work

Restricting the class of schedulers has been observed to be important in obtaining compositional reasoning principles [14], and in correctly analyzing security protocols and distributed algorithms [13, 12, 6, 15, 11]. The schedulers considered in these papers are very similar to the class of view consistent schedulers that we consider. In [14], the processes are assumed to run synchronously, and thus the scheduler is the composition of local schedulers that resolve nondeterminism based on local views; there is no global scheduler. In [12, 6], the nondeterministic choices are broken into tasks. A task scheduler chooses the task, and this choice is assumed to be *oblivious* of the actual computation, and local scheduler picks the actual transition within the task by looking at the local state. The task scheduler can be seen as our global scheduler; however, the difference is that our global schedulers are not oblivious. Finally, in [11], the authors don't restrict attention to a specific class of scheduler but rather develop a process calculus within which the schedulers can be specified. All these papers, are

primarily interested in defining clean compositional semantics, and do not consider the model checking problem per se. A closely related class of schedulers called *distributed schedulers* is considered in [18, 16, 17], where the problem of model checking safety properties against any threshold is shown to be undecidable. However, distributed schedulers are different than view consistent schedulers that we consider here — in a distributed scheduler, a local scheduler of process i is completely oblivious of steps in which process i did not get scheduled, whereas in view consistent schedulers, it is aware that some other process was scheduled. This difference is manifested in the fact that $\mathbb{P} \models_{>0} \mathsf{Spec}$ for safety specifications is undecidable for distributed schedulers [16, 17], whereas it is open for view consistent schedulers. Furthermore, no decidability results are presented in [18, 16, 17].

As indicated in the introduction, the model checking problem and its complexity for the related model of Partially Observable MDPs (POMDP) has been investigated in earlier works [13, 20, 2, 10]. A POMDP $\mathbb{P}$ can be seen as a special case of a concurrent program with two processes under view consistent semantics as follows. The POMDP itself is process $\mathcal{P}_1$, and the second process (say $\mathcal{P}_2$) plays the role of "scheduling" the next transition of $\mathcal{P}_1$. They share 3 variables: state that stores the partial state of $\mathcal{P}_1$ that is visible outside, trans that is used by $\mathcal{P}_2$ to inform $\mathcal{P}_1$ what the next transition should be, and turn which is used by the processes to alternate taking turns. In each "round", $\mathcal{P}_2$ first picks $\mathcal{P}_1$'s next transition, and then $\mathcal{P}_1$ "executes" that transition; observe, that $\mathcal{P}_1$ is a purely probabilistic process, and all the nondeterminism has been deferred to $\mathcal{P}_2$. Under view consistent schedulers, the two processes $\mathcal{P}_1$ and $\mathcal{P}_2$ are "equivalent" to the POMDP. Also decision problems for any programs whose all, except possibly one processes are purely probabilistic, can in turn be shown to be "equivalent" to a POMDP of size exponential in the length of the program (see Lemma 3.9). This relationship is exploited by us to prove some upper bounds.

Similarly, the "equivalence" between decision problems on probabilistic automata on infinite strings and decision problems on programs under "locally Markovian" schedulers is exploited to obtain the undecidability results using the results of [2, 7]. This equivalence is also exploited to obtain upper bounds for checking $\mathbb{P} \models^{lm}_{=1} \mathsf{Spec}$ when Spec is deterministic, and checking $\mathbb{P} \models^{lm}_{>0} \mathsf{Spec}$ when Spec is a safety property.

2 Definitions

2.1 Preliminaries

The powerset of any set A will be denoted by 2^A. Given any set Σ, Σ^+ will denote the set of nonempty finite words over Σ and Σ^ω the set of infinite words over Σ. Given a word $\alpha \in \Sigma^+ \cup \Sigma^\omega$, we will denote the length of α by $length(\alpha)$ (length of α is ω for $\alpha \in \Sigma^\omega$). We assume that the reader is familiar with basic measure theory. We will also assume familiarity with finite automata on infinite strings and Partially Observable Markov Decision Processes (POMDP).

2.1.1 Probabilistic Automata

We recall the definition of probabilistic Büchi automata (PBA)s [3]. Informally, a PBA is like a deterministic Büchi automata except that the transition function from a state on a given input is described as a probability distribution that determines the probability of the next state. Formally, a PBA over a finite alphabet Δ is a tuple $\mathcal{B} = (Q, q_s, Q_f, \delta)$ where Q is a finite set of *states*, $q_s \in Q$ is the *initial state*, $Q_f \subseteq Q$ is the set of *accepting/final states*, and $\delta : Q \times \Delta \times Q \to [0, 1]$ is the *transition relation* such that for all $q \in Q$ and $a \in \Delta$,

$\sum_{q' \in Q} \delta(q, a, q') = 1$. For this paper, we assume that $\delta(q, a, q')$ is a rational number for all $q, q' \in Q$ and $a \in \Delta$.

Intuitively, the PBA $\mathcal{B}$ starts in the initial state q_s and if after reading $a_0, a_1 \ldots, a_i$ results in state q, it moves to state q' with probability $\delta(q, a_{i+1}, q')$ on symbol a_{i+1}. Given a word $\alpha \in \Delta^\omega$, $\mathcal{B}$ can be thought of as an infinite-state Markov chain which gives rise to the standard σ-algebra defined using cylinders and the standard probability measure on Markov chains [25, 21]. We denote this measure by $\mu_{\alpha,\mathcal{B}}$. A *run* of $\mathcal{B}$ is an infinite sequence $\rho \in Q^\omega$. A run ρ is *accepting* if ρ satisfies the Büchi acceptance condition, *i.e.*, $\rho[i] \in Q_f$ for infinitely many i.

The set of accepting runs is measurable. Given α, the measure of the set of accepting runs will be denoted by $\mu^{acc}_{\mathcal{B},\alpha}$ and is said to be the *probability of accepting* α. Given $x \in [0, 1]$ and $\triangleright \in \{>, =, \geq\}$, we let $\mathcal{L}_{\triangleright x}(\mathcal{B}) = \{\alpha \in \Delta^\omega \mid \mu^{acc}_{\mathcal{B},\alpha} \triangleright x\}$.

We identify one useful syntactic restriction of PBAs, called *finite probabilistic monitors* (FPM)s [7]. In a FPM, all the states are accepting except a special absorbing *reject* state (a state q_r is said to be *absorbing* if $\delta(q_r, a, q_r) = 1$ for each input $a \in \Delta$). By using a set of Rabin pairs instead of a set of final states, we can define *Probabilistic Rabin automata* (PRAs).

2.2 Programs

We will denote the set of Boolean expressions over Boolean variables V by $\mathsf{BEXP}(V)$. The value of a Boolean expression BEXP under a truth assignment $s : V \to \{0, 1\}$ will be denoted by $[\![\mathsf{Bexp}]\!]_s$. We use 2^V to denote the set of assignments on V. An *update* to variables in V is a set of assignments of the form $x := \mathsf{Bexp}$, such that each variable appears the left hand side of at most one assignment in the set. An update A defines a function $\mathsf{app}_A : 2^V \to 2^V$ as follows: if $x := \mathsf{Bexp} \in A$ then $\mathsf{app}_A(s)(x) = [\![\mathsf{Bexp}]\!]_s$, and if x is not on the left hand side of any assignment in A then $\mathsf{app}_A(s)(x) = s(x)$. We say that s' is obtained by applying the update A to s if $\mathsf{app}_A(s) = s'$.

A probabilistic concurrent program $\mathbb{P}$ with n processes is a tuple $(V, s_0, (V_1, \mathcal{P}_1), ..., (V_n, \mathcal{P}_n))$. Here V_i is a finite set of Boolean variables that process i reads and writes to, with $V = \cup_{i=1}^n V_i$ being the *set of program variables*. $s_0 \in 2^V$ is the *initial state* of the program, and $\mathcal{P}_i$ is a finite set of transitions of process i defined as follows. Each transition τ of process $\mathcal{P}_i$ is of the form $(C,\ p_1 : A_1, p_2 : A_2, ..., p_k : A_k)$ where C is a Boolean expression on V_i, and $(p_1, ..., p_k)$ is a sequence of nonzero rational probabilities that add up to 1 and $A_1, ..., A_k$ are *updates* such that all the variables appearing (on the left hand side or right hand side of an assignment) in A_j are in V_i. For any i, j, we say that processes i, j communicate if $V_i \cap V_j \neq \emptyset$. For any i, $1 \leq i \leq n$, let $L_i = V_i - \cup_{j \neq i} V_j$, namely, the set of variables of $\mathcal{P}_i$ that are not visible to any other process. The variables in L_i are said to be *local variables* of process i. We will also assume, without loss of generality, that each process has at least one variable — a process i without any variables can be modeled in our framework as a process with one local variable whose value remains constant.

The *states* of $\mathbb{P}$ will be 2^V. Let $\tau = (C,\ p_1 : A_1, p_2 : A_2, ..., p_k : A_k)$ be a transition of a process $\mathcal{P}_i$. We say that τ is *enabled* in state s if C is satisfied in s. The process $\mathcal{P}_i$ is said to be *deterministic (or purely probabilistic)* if for each state s, there is at most one transition of $\mathcal{P}_i$ enabled in s. Assume that τ is enabled in s. If the transition τ is *executed* in state s, then one of the updates $A_1, ..., A_k$ is chosen, with the probability distribution given by $p_1, ..., p_k$ and applied to the state s. Let t_i be the state obtained by performing the update A_i to the state s. We say that the probability that the next state is t_i is p_i when transition τ is executed in state s. We assume that for each state s, there is some process $\mathcal{P}_i$ such that

some transition of $\mathcal{P}_i$ is enabled in s. For any state s and process index i, $1 \leq i \leq n$, we let $s|i$ denote the restriction of s to the variables in V_i. Intuitively, $s|i$ denotes the part of the state that is visible to process i, i.e., the local state.

A program is interpreted using schedulers which, depending on the history, resolve nondeterminism by assigning which of the enabled actions is fired in a given state.

Classes of Schedulers. Let $\mathbb{P}$ be a program with n processes $\mathcal{P}_1, \ldots \mathcal{P}_n$. Let 2^V be the set of states of $\mathbb{P}$ and $Trans$ be the set of transitions of $\mathbb{P}$. A *history* is an element of $(2^V)^+$. Given a history $h = t_0 ... t_m$, we define $last(h)$ to be the state t_m and $length(h)$ to be $m+1$. Given a process index i, $0 \leq i \leq n$, we define $h|i$ to be the word $(t_0|i)(t_1|i)...(t_m|i)$. Intuitively, $h|i$ denotes the view of process i in h.

A scheduler $\eta : (2^V)^+ \to Trans$ is a function that associates, with each history of a program $\mathbb{P}$, a transition τ of some process of $\mathbb{P}$ that is enabled in the last state of the history. We say that a scheduler η is *view consistent* if the following property holds for every pair of histories h, h' and every process index $1 \leq i \leq n$: if $\eta(h)$, $\eta(h')$ are both transitions of process i and $h|i = h'|i$ then $\eta(h) = \eta(h')$. Intuitively, view consistency requires that the transition of a process, chosen by the scheduler, should depend only on the view of the process; that is, the nondeterminism within a process is resolved based purely on process' view of the computation history. Note that the above condition does not prevent the scheduler from choosing transitions of different processes for h and h'.

We say that η is *locally Markovian* (or *locally step dependent*) if the following property holds for every pair of histories h, h' and every process i: if $\eta(h)$, $\eta(h')$ are both transitions of process i, $length(h) = length(h')$, and $last(h)|i = last(h')|i$, then $\eta(h) = \eta'(h)$. Note that in this case, the transition scheduled should only depend on the length of the history and the current local state of the process. Observe that every locally Markovian scheduler is also view consistent.

Computations. In presence of a scheduler η, a program $\mathbb{P}$ with 2^V as set of states can be thought of as an infinite-state Markov chain which gives rise to the standard σ-algebra on $(2^V)^\omega$ and the standard probability measure [25, 21]. We will denote this Markov chain as $\mathcal{M}_{\mathbb{P},\eta}$ and the standard probability measure generated as $\mu_{\mathcal{M}_{\mathbb{P},\eta}}$. The set $(2^V)^\omega$ shall be called the set of paths of $\mathcal{M}_{\mathbb{P},\eta}$.

Let $\mathcal{A}$ be a Büchi automaton with 2^V (the set of states of $\mathbb{P}$) as its input alphabet. We say that $\mathcal{A}$ accepts an infinite path $\rho \in (2^V)^\omega$ of $\mathcal{M}_{\mathbb{P},\eta}$, if it accepts the infinite sequence ρ. Let $\mathcal{L}(\mathcal{A})$ be the language accepted by $\mathcal{A}$. Now $\mathcal{L}(\mathcal{A})$ is a measurable set in the space of paths defined by $\mathcal{M}_{\mathbb{P},\eta}$ and we call the measure of $\mathcal{L}(\mathcal{A})$, $\mu_{\mathcal{M}_{\mathbb{P},\eta}}(\mathcal{L}(\mathcal{A}))$, to be the *probability of acceptance of* $\mathcal{M}_{\mathbb{P},\eta}$. Given a rational number x and $\triangleright \in \{>, =, \geq\}$, we shall write $\mathbb{P}, \eta \models_{\triangleright x} \mathcal{A}$ if $\mu_{\mathcal{M}_{\mathbb{P},\eta}}(\mathcal{L}(\mathcal{A})) \triangleright x$. The automaton $\mathcal{A}$ will be henceforth called a *specification automaton*.

Predicate automaton. It is often useful to present the specification automaton $\mathcal{A}$ succinctly in the following fashion. A *predicate automaton* Spec is a tuple $(V, Q, q_s, Q_f, \to)$ where V is a finite set of boolean variables, Q is a finite set of *states*, $q_s \in Q$ is the *initial state*, $Q_f \subseteq Q$ is the set of *final states* and $\to \subseteq Q \times \mathsf{BEXP}(V) \times Q$ is a *finite* set of *predicate transitions*. Given a predicate automaton Spec, we define a specification automaton $[\![\mathsf{Spec}]\!]$ as follows: $[\![\mathsf{Spec}]\!] = (2^V, Q, q_s, Q_f, \delta)$ where $(q, s, q') \in \delta$ iff there is a predicate transition $(q, \mathsf{Bexp}, q') \in \to$ such that s satisfies Bexp. Please note given any specification (Büchi) automaton $\mathcal{A}$, there is a predicate automaton Spec such that $[\![\mathsf{Spec}]\!] = \mathcal{A}$. Furthermore, we will say that Spec is a deterministic predicate automaton (respectively safety) if $[\![\mathsf{Spec}]\!]$ is a deterministic automaton (respectively safety automaton). Whenever convenient, we will often confuse Spec with $[\![\mathsf{Spec}]\!]$. Given any history $h \in \Sigma^*$ let $\mathsf{Spec}(h)$ be the state that

	ω-REGULAR SPEC	DETERMINISTIC SPEC	SAFETY SPEC
$\mathbb{P} \models^{vc}_{=1}$ Spec	Undecidable	?[b]	**PSPACE**-complete
$\mathbb{P} \models^{lm}_{=1}$ Spec	Undecidable	**EXPSPACE**-complete	**PSPACE**-complete
$\mathbb{P} \models_{=1}$ Spec	**EXPTIME**-complete[a]	**EXPTIME**-complete	**PSPACE**-complete
$\mathbb{P} \models^{vc}_{>0}$ Spec	Undecidable	Undecidable	?[b]
$\mathbb{P} \models^{lm}_{>0}$ Spec	Undecidable	Undecidable	**EXPSPACE**-complete
$\mathbb{P} \models_{>0}$ Spec	**EXPTIME**-complete[a]	**EXPTIME**-complete	**EXPTIME**-complete
$\mathbb{P} \models^{vc}_{\triangleright x}$ Spec	Undecidable	Undecidable	Undecidable
$\mathbb{P} \models^{lm}_{\triangleright x}$ Spec	Undecidable	Undecidable	Undecidable
$\mathbb{P} \models_{\triangleright x}$ Spec	**EXPTIME**-complete[a]	**EXPTIME**-complete	**EXPTIME**-complete

Figure 1 Summary of complexity results. For entries with superscript a, we assume that Spec is given as a deterministic predicate Rabin automaton. For entries with superscript b, the problem becomes 2-**EXPTIME**-complete if either (i) all but one processes of $\mathbb{P}$ are deterministic (purely probabilistic), or (ii) if the processes communicate through global variables and are mutually exclusive, and Spec is on the shared state.

$[\![\mathsf{Spec}]\!]$ is in after reading h from its initial state.

Similar to the predicate automaton, we can define a *predicate Rabin automaton*, in which instead of using a set of final states, we use a set of Rabin pairs. As in the case of predicate automaton, a predicate Rabin automaton gives rise to a Rabin automaton.

Verification. Given a rational number x and $\triangleright \in \{\geq, =, >\}$, we will write $\mathbb{P} \models_{\triangleright x}$ Spec if for every scheduler η, we have that $\mathbb{P}, \eta \models_{\triangleright x} [\![\mathsf{Spec}]\!]$. Similarly, we write $\mathbb{P} \models^{vc}_{\triangleright x}$ Spec ($\mathbb{P} \models^{lm}_{\triangleright x}$ Spec, respectively) if for every view consistent scheduler η (locally Markovian scheduler, respectively), $\mathbb{P}, \eta \models_{\triangleright x} [\![\mathsf{Spec}]\!]$. Thus, the verification problem we consider is one where given $\mathbb{P}$, Spec, rational number $x \in [0, 1]$, and $\triangleright \in \{\geq, =, >\}$ as input, we want to determine if $\mathbb{P} \models_{\triangleright x}$ Spec (or $\mathbb{P} \models^{vc}_{\triangleright x}$ Spec or $\mathbb{P} \models^{lm}_{\triangleright x}$ Spec). The predicate automaton Spec is often called the *specification*.

3 Complexity and decidability for general programs

In this section, we present results on the decidability and complexity of the verification problems defined in Section 2. Our results are summarized in Figure 1. We will now state the results. The missing proofs as well as all the proofs/results of all schedulers can be found in [9].

3.1 Undecidability

We start by establishing the undecidability of the model checking problem for concurrent programs in a variety of settings.

▶ **Theorem 3.1.** *Given a program $\mathbb{P}$ and a predicate automaton* Spec*, the following problems are undecidable.*

(a) Determining if $\mathbb{P} \models^{vc}_{=1}$ Spec and $\mathbb{P} \models^{lm}_{=1}$ Spec.

(b) Determining if $\mathbb{P} \models^{vc}_{>0}$ Spec and $\mathbb{P} \models^{lm}_{>0}$ Spec, even when Spec *is a deterministic specification.*

(c) Given a rational $x \in (0, 1)$ and $\triangleright \in \{>, \geq\}$, determining if $\mathbb{P} \models^{vc}_{\triangleright x}$ Spec and $\mathbb{P} \models^{lm}_{\triangleright x}$ Spec, even when Spec *is a safety specification.*

The above undecidability results continue to hold even if $\mathbb{P}$ is restricted to be a program consisting of two processes, one of which is purely nondeterministic (no probabilistic transitions) and the other is purely probabilistic (no nondeterministic transitions).

Theorem 3.1 is proved as follows: for part (a) we reduce the problem of checking if a given PRA accepts every input with probability 1; for part (b) we reduce the problem of checking if a given PBA accepts every input with probability > 0; and for part (c) we reduce the problem of checking if a given FPM accepts every input with probability $\triangleright x$.

For a program $\mathbb{P}$, the observations in Theorem 3.1, leave open the decidability of the following questions.

(a) if Spec is a safety specification, check if $\mathbb{P} \models^{vc}_{=1}$ Spec (or check if $\mathbb{P} \models^{lm}_{=1}$ Spec)?
(b) if Spec is a safety specification, check if $\mathbb{P} \models^{vc}_{>0}$ Spec (or check if $\mathbb{P} \models^{lm}_{>0}$ Spec)?
(c) if Spec is a deterministic specification, check if $\mathbb{P} \models^{vc}_{=1}$ Spec (or check if $\mathbb{P} \models^{lm}_{=1}$ Spec)?

We address these questions in the forthcoming sections. The decidability of (a) is shown in Theorem 3.2 in Section 3.2. The problems in (b) and (c) are also shown to be decidable for locally Markovian schedulers, in Section 3.2; these problems remain open for the case of view consistent schedulers. However, in Section 3.3, we show that these problems are decidable for two special classes of programs.

▶ Remark. Distributed schedulers, introduced in [18] and further studied in [16, 17], are very similar to view consistent schedulers that we consider here. However, there is one important, subtle difference between them. In a view consistent scheduler, the local scheduler of process i is aware of both the steps when process i was scheduled and those when it was not scheduled; in distributed schedulers the local scheduler is only aware of the steps when it was scheduled. Thus, the undecidability results presented here do not follow from [18, 16, 17]. Moreover, in [16, 17], the problem of checking if $\mathbb{P} \models_{>0}$ Spec for safety specifications Spec under all distributed schedulers is shown to be undecidable; however, that proof does not extend to view consistent schedulers and this problem for view consistent schedulers (as stated in the discussion above) is open.

3.2 Decidability results for locally Markovian semantics

We begin by establishing the decidability of checking if the measure of computations accepted by a safety specification, under every scheduler in a class $\mathcal{C}$, is 1. We, in fact, show that for any of the three classes of schedulers that we consider, this problem is **PSPACE**-complete.

▶ **Proposition 3.2.** *Given a program $\mathbb{P}$ and a safety specification* Spec*, the following problems are* **PSPACE***-complete: determining if $\mathbb{P} \models^{vc}_{=1}$* Spec*, if $\mathbb{P} \models^{vc}_{=1}$* Spec *and if $\mathbb{P} \models^{lm}_{=1}$* Spec*.*

We now establish that the problems of determining if $\mathbb{P} \models^{lm}_{>0}$ Spec when Spec is a safety specification, and of determining if $\mathbb{P} \models^{lm}_{=1}$ Spec when Spec is deterministic are **EXPSPACE**-complete. We begin by defining a special class of locally Markovian schedulers that we call Spec-*determined*. These are schedulers that are required to choose the same transition after equal length histories h and h' that end in the same state, if the state reached by the specification $[\![\mathsf{Spec}]\!]$ after h (namely, $\mathsf{Spec}(h)$) is the same as $\mathsf{Spec}(h')$.

▶ **Definition 3.3.** Let $\mathbb{P}$ be a program with n processes, and let Σ be the set of states of $\mathbb{P}$ and $Trans$ be the set of transitions of $\mathbb{P}$. Let Spec be a deterministic specification with Q as the set of states. We say that a locally Markovian scheduler $\eta : \Sigma^+ \to Trans$ is Spec-*determined* if for any pair of histories $h, h' \in \Sigma^+$ such that $length(h) = length(h')$, $\mathsf{Spec}(h) = \mathsf{Spec}(h')$ and $last(h) = last(h')$, we have that $\eta(h) = \eta(h')$.

The reason for considering Spec-determined schedulers is because we can show that for the problems of verifying safety with non-zero probability and verifying deterministic specifications with probability 1, we can restrict our attention to Spec-determined schedulers. This is the content of the next proposition.

▶ **Proposition 3.4.** *For any program* $\mathbb{P}$ *and safety specification* Spec, $\mathbb{P} \models^{lm}_{>0}$ Spec *iff for any* Spec*-determined locally Markovian scheduler* η*;* $\mathbb{P}, \eta \models_{>0}$ Spec. *For deterministic specification* Spec, $\mathbb{P} \models^{lm}_{=1}$ Spec *iff for all* Spec*-determined locally Markovian schedulers* η*;* $\mathbb{P}, \eta \models_{=1}$ Spec.

We need one more definition.

▶ **Definition 3.5.** Let $\mathbb{P}$ be a program with n processes with V as the set of variables and $Trans$ as the set of transitions. Let Spec be a deterministic specification. We say a function $g : Q \times 2^V \to Trans$ is Spec*-determined* and *locally consistent* if for all $q \in Q$ and $s \in 2^V$, $g((q, s))$ is enabled in s; and $g((q_1, s_1)) = g((q_2, s_2))$ whenever $(q_1, s_1), (q_2, s_2) \in Q \times 2^V$ are such that $g(q_1, s_1), g(q_2, s_2)$ belong to the same process $\mathcal{P}_i$ and $s_1|i = s_2|i$. The set of Spec-determined and locally consistent functions of program $\mathbb{P}$ shall be denoted as $\mathsf{Loc}(\mathbb{P}, \mathsf{Spec})$.

Given a deterministic specification Spec, it is easy to see that there is a bijection between the set of Spec-determined locally Markovian schedulers of a program $\mathbb{P}$ and $(\mathsf{Loc}(\mathbb{P}, \mathsf{Spec}))^\omega$, the set of infinite sequences over $(\mathsf{Loc}(\mathbb{P}, \mathsf{Spec}))$. We call this function $\mathsf{Loc}_{\mathbb{P},\mathsf{Spec}}$. The key technical idea exploited in our model checking algorithm is the following. Given a program $\mathbb{P}$ and a specification Spec, one can construct a PBA $\mathcal{B}$ that accepts a word $\mathsf{Loc}_{\mathbb{P},\mathsf{Spec}}(\eta)$ with the same probability as the computation of $\mathcal{P}$ under scheduler η satisfies Spec.

▶ **Lemma 3.6.** *Given a program* $\mathbb{P}$ *and a deterministic specification* Spec*, let* Δ *be* $\mathsf{Loc}(\mathbb{P}, \mathsf{Spec})$*, the set of* Spec*-determined locally consistent functions. There is a* PBA $\mathcal{B}$ *on input alphabet* Δ *such that the following hold–*

- *The number of states of* $\mathcal{B}$ *is exponential in the size of* $\mathbb{P}$ *and* Spec*.*
- *For any* Spec*-determined locally Markovian scheduler* η*, the probability that the computation* $\mathcal{M}_{\mathbb{P},\eta}$ *satisfies* Spec *is the probability of* $\mathsf{Loc}_{\mathbb{P},\mathsf{Spec}}(\eta)$ *being accepted by* $\mathcal{B}$*.*
- $\mathcal{B}$ *can be taken to be a* FPM *if* Spec *is a safety specification.*

We have the following theorem.

▶ **Theorem 3.7.** *Given a program* $\mathbb{P}$ *and a deterministic specification* Spec *the problem of determining if* $\mathbb{P} \models^{lm}_{=1}$ Spec *is* **EXPSPACE***-complete. If* Spec *is a safety specification, then the problem of determining if* $\mathbb{P} \models^{lm}_{>0}$ Spec *is also* **EXPSPACE***-complete.*

We had shown in [7, 8] that given a PBA $\mathcal{B}$, the problem of checking whether all words are accepted with probability 1 is in **PSPACE**. We had also shown in [7] that given a FPM $\mathcal{M}$, the problem of checking whether all words are accepted with probability > 0 is in **PSPACE**. In light of Lemma 3.6, this immediately implies that the problems of determining whether a program satisfies a deterministic specification with probability 1 and whether a program satisfies a safety specification with nonzero probability are decidable. However, note that as the input alphabet constructed in Lemma 3.6 is doubly-exponential in the size of the input, the straightforward application of the results in [7, 8] do not lead to inclusion in **EXPSPACE**. The inclusion in **EXPSPACE** is achieved by a careful examination of algorithms given in [7, 8] and running the algorithm without explicitly constructing the PBA.

3.3 Decidability results for view consistent semantics

Proposition 3.2 already establishes that checking whether every computation of a program $\mathbb{P}$ generated by a view consistent scheduler satisfies a safety specification with probability 1 is **PSPACE**-complete. We now consider the remaining questions, namely, checking whether $\mathbb{P} \models^{vc}_{>0}$ Spec when Spec is a safety property, and checking whether $\mathbb{P} \models^{vc}_{=1}$ Spec when Spec is a deterministic specification. While the decidability of these problems is open, we prove decidability for special classes of programs $\mathbb{P}$. First we show that these problems are **2-EXPTIME**-complete when all processes of $\mathbb{P}$, except possibly one, are deterministic.

▶ **Theorem 3.8.** *Given a program $\mathbb{P}$ and a deterministic specification* Spec *such that all processes of $\mathbb{P}$ except one are deterministic, the problem of checking if $\mathbb{P} \models^{vc}_{=1}$ Spec is* **2-EXPTIME***-complete. Given a safety specification* Spec*, the problem of checking if $\mathbb{P} \models^{vc}_{>0}$ Spec is also* **2-EXPTIME***-complete.*

The main idea behind the proof of **2-EXPTIME** membership is to reduce it to model checking POMDPs. The following is proved in [9].

▶ **Lemma 3.9.** *Given a program $\mathbb{P}$ and a deterministic specification (safety specification, respectively)* Spec *such that all processes of $\mathbb{P}$ except one are deterministic, there is a* POMDP *$\mathcal{M}$ and a subset Q of states of $\mathcal{M}$ such that $\mathbb{P} \models^{vc}_{=1}$ Spec ($\mathbb{P} \models^{vc}_{>0}$ Spec, respectively) iff under every observation based scheduler, the measure of paths of $\mathcal{M}$ that visit Q infinitely often is 1 (> 0, respectively).*

The second special class of programs that we consider are the following. Processes in a program are said to communicate through *global variables* if every pair of processes share the same set of variables. We say that processes in $\mathbb{P}$ are *mutually exclusive* if in every state the transitions of only one process are enabled. A deterministic specification Spec is said to be *on the shared state* if whenever (q, Bexp, q') is a transition of Spec, Bexp evaluates to the same value for any two program states in which the global variables take the same value.

▶ **Theorem 3.10.** *Given a program $\mathbb{P}$ where the processes communicate through global variables and are mutually exclusive, and a deterministic specification (safety specification, respectively)* Spec *on the shared state, the problem of checking $\mathbb{P} \models^{vc}_{=1}$ Spec ($\mathbb{P} \models^{vc}_{>0}$ Spec, respectively) is* **2-EXPTIME***-complete.*

The proof of the **2-EXPTIME**-decidability relies on showing that if there is a scheduler η such that $\mathbb{P}, \eta \models_{<1}$ Spec ($\mathbb{P}, \eta \models_{=0}$ Spec) for a deterministic specification on the shared state (safety specification) then there is a "periodic" scheduler η' that witnesses the same fact. The model checking algorithm searches for such a periodic scheduler by reducing it to μ-calculus model checking on a finite (doubly exponentially sized) bi-partite graph $\mathcal{G}$. We illustrate the construction of the graph $\mathcal{G}$ for the case when Spec is a deterministic specification.

Let $\mathbb{P}$ be a program where the processes communicate through global variables and are mutually exclusive, and Spec be a deterministic specification on the shared state. We start by some definitions. Assume that $\mathbb{P}$ has n processes, V is the set of shared variables and V_i is the set of local variables of process i. It is easy to see that the set of states of $\mathbb{P}$ can be taken to be $2^V \times 2^{V_1} \times \ldots \times 2^{V_n}$. We henceforth refer to this set as $States(\mathbb{P})$. If Q is the set of states of Spec then the set $Q \times States(\mathbb{P})$ is said to be the set of *extended states* and will be referred to by $EStates(\mathsf{Spec}, \mathbb{P})$. An extended state $es = (q, s)$ is said to be *feasible* if there is a history h of $\mathbb{P}$ such that the measure of h is > 0, $h(0)$ is the initial state of $\mathbb{P}$, $last(h) = s$ and $\mathsf{Spec}(h) = q$. Let $\pi_{\mathbb{P}} : EStates(\mathsf{Spec}, \mathbb{P}) \to States(\mathbb{P})$ be the map $\pi_{\mathbb{P}}((q, s)) = s$. Given $es = (q, s) \in EStates(\mathsf{Spec}, \mathbb{P})$ and a Spec-determined and locally consistent function g (see Definition 3.5), let $\mathsf{succ}_g(es) = \{(q', s') \mid (q, s, q')$ is a transition of $[\![\mathsf{Spec}]\!]$ & s' is obtained with nonzero probability when $g((q, s))$ is executed in $s\}$. Given a set $U \subseteq EStates(\mathsf{Spec}, \mathbb{P})$, let $\mathsf{succ}_g(U) = \cup_{es \in U} \mathsf{succ}_g(es)$.

A set of states $S \subseteq States(\mathbb{P})$ is said to be *closed* if $S = \{v\} \times S_1 \times \ldots \times S_n$ for some $v \in 2^V$ and $S_i \in 2^{V_i}$. A set $U \subseteq EStates(\mathsf{Spec}, \mathbb{P})$ is said to be an *extended closed set* if $\pi_{\mathbb{P}}(U)$ is closed. We show in [9] that for any extended closed set U, $\mathsf{succ}_g(U)$ can be partitioned into a union of *disconnected* extended closed sets. Two extended closed sets U_1 and U_2 are said to be disconnected if for each $es_1 \in U_1$ and $es_2 \in U_2$ and each process i, $\pi_{\mathbb{P}}(es_1)|i \neq \pi_{\mathbb{P}}(es_2)|i$. We shall call these disconnected sets *components* of $\mathsf{succ}_g(U)$.

The bi-partite graph $\mathcal{G}$ consists of 2 partitions, W_1 and W_2. The set W_1 is the set of extended closed sets. W_2 is the set of pairs (U, g) where U is an extended closed set and g a Spec-determined and locally consistent function. There is an edge from $U \in W_1$ to $(U', g) \in W_2$ iff $U = U'$. There is an edge from (U, g) to U' iff U' is a component of $\mathsf{succ}_g(U)$. We convert $\mathcal{G}$ into a Kripke structure by labeling each node of $\mathcal{G}$ by a special proposition F or its negation $\neg F$ as follows. A node U in W_1 is labeled with F iff there is an extended state $es = (q, s) \in U$ such that q is a final state of Spec. Every other node of W_1 and each node of W_2 is labeled by $\neg F$. We denote the resulting Kripke structure by $\mathcal{G}(\mathsf{Spec}, \mathbb{P})$. The 2-**EXPTIME**-decidability of checking if $\mathbb{P} \models^{vc}_{=1} \mathsf{Spec}$ follows from the following lemma shown in [9].

▶ **Lemma 3.11.** Given a program $\mathbb{P}$ where the processes communicate through global variables and are mutually exclusive, and a deterministic specification Spec on the shared state, let $\mathcal{G}(\mathsf{Spec}, \mathbb{P})$ be the Kripke structure obtained as described above. There is a view consistent scheduler η such that $\mathcal{M}_{\mathbb{P},\eta}$ satisfies the specification Spec with probability < 1 iff there is a feasible extended state es such that the node $\{es\}$ in $\mathcal{G}(\mathsf{Spec}, \mathbb{P})$ satisfies the modal μ-calculus formula $f = \nu X(\neg F \wedge \Diamond\Box X)$ where $\Diamond$ and $\Box$ are the existential and universal "nexttime" operators and νX is the greatest fixpoint operator.

4 Conclusions

Randomization and nondeterminism play an important role in concurrent processes, and in this paper we showed that to get accurate verification results, one needs to consider restricted classes of schedulers. Tight complexity bounds for verifying linear time properties under restricted classes of schedulers were established. Our complexity results confirm observations made in [13] that restricting the class of schedulers makes the verification problem more difficult.

The global schedulers we consider can observe all the variables in the program. There may be situations when we want to restrict the power of the global scheduler as well. However, this is easily captured in our setting by adding a new process P_{new} that can only see a part of the state that is visible to the restricted global scheduler. This new process will execute odd step (ensured by adding a new turn variable), and will pick the process to schedule based on the partial state it sees.

View consistent and locally Markovian schedulers are just some of the classes that might be useful in the concurrent setting. One natural class of schedulers we have not explicitly mentioned in this paper are memoryless schedulers, where the choice made by the scheduler depends only on the current state and not on the history. It is easy to see that the verification problems are in co-**NEXPTIME**— guess the scheduler that violates the property, and check that under the scheduler the system (which is now a finite state MDP) violates the property. The verification problems are also likely to be co-**NEXPTIME**-hard based on observations made in [13]; once again the blowup in complexity being explained by the state space explosion problem. One restriction of the schedulers we consider here is that the local scheduler for a process is "aware" of the fact that other processes were scheduled. This may or may not be reasonable in some settings. In the future we would like to expand the current investigations to other useful classes of schedulers.

Acknowledgements. The authors would like to thank anonymous referees for sending pointers to [18, 16, 17]. A. Prasad Sistla was supported by NSF-0720525, NSF CCF-0916438, NSF CNS-1035914 and Mahesh Viswanathan was supported by NSF CCF 0448178, NSF CCF 1016989, and NSF CNS 1016791.

References

1 PRISM — Probabilistic Symbolic Model Checker. `http://www.prismmodelchecker.org`.
2 C. Baier, N. Bertrand, and M. Größer. On decision problems for probabilistic Büchi automata. In *Proceedings of FoSSaCS*, pages 287–301, 2008.
3 C. Baier and M. Grö„er. Recognizing ω-regular languages with probabilistic automata. In *Proceedings of LICS*, pages 137–146, 2005.
4 C. Baier, B. Haverkrot, H. Hermanns, and J.-P. Katoen. Efficient computation of time-bounded reachability probabilisties in uniform continuous-time Markov decision processes. In *Proceedings of TACAS*, pages 61–76, 2004.
5 A. Bianco and L. de Alfaro. Model checking of probabalistic and nondeterministic systems. In *Proceedings of FSTTCS*, pages 499–513, 1995.
6 R. Canetti, L. Cheung, D. Kaynar, M. Liskov, N. Lynch, P. Pereira, and R. Segala. Task-Structured Probabilistic I/O Automata. In *Workshop on Discrete Event Systems*, 2006.
7 R. Chadha, A. P. Sistla, and M. Viswanathan. On the expressiveness and complexity of randomization in finite state monitors. *Journal of the ACM*, 56(5), 2009.
8 R. Chadha, A. P. Sistla, and M. Viswanathan. Power of randomization in automata on infinite strings. In *Proceedings of CONCUR*, pages 229–243, 2009.
9 R. Chadha, A. P. Sistla, and M. Viswanathan. Model checking concurrent programs with nondeterminism and randomization. Technical Report LSV-10-15, LSV, ENS Cachan, 2010.
10 K. Chatterjee, L. Doyen, and T. Henzinger. Qualitative Analysis of Partially-observed Markov Decision Processes. *CoRR*, abs/0909.1645, 2009.
11 K. Chatzikokolakis and C. Palamidessi. Making Random Choices Invisible to the Scheduler. *Information and Computation*, 2010, to appear.
12 L. Cheung. *Reconciling Nondeterministic and Probabilistic Choices*. PhD thesis, Radboud University of Nijmegen, 2006.
13 L. de Alfaro. The Verification of Probabilistic Systems under Memoryless Partial Information Policies is Hard. In *Proceedings of PROBMIV*, 1999.
14 L. de Alfaro, T. Henzinger, and R. Jhala. Compositional methods for probabilistic systems. In *Proceedings of CONCUR*, pages 351–365, 2001.
15 F.D. Garcia, P. van Rossum, and A. Sokolova. Probabilistic Anonymity and Admissible Schedulers. *CoRR*, abs/0706.1019, 2007.
16 S. Giro. Undecidability results for distributed probabilistic systems. In *Proceedings of SBMF*, pages 220–235, 2009.
17 S. Giro. *On the automatic verification of Distributed Probabilistic Automata with Partial Information*. PhD thesis, Universidad Nacional de Córdoba, 2010.
18 S. Giro and P.R. D'Argenio. Quantitative model checking revisited: Neither decidable nor approximable. In *Proceedings of FORMATS*, pages 179–194, 2007.
19 S. Goldwasser and S. Micali. Probabilistic encryption and how to play mental poker keeping secret all partial information. In *Proceedings of STOC*, pages 365–377, 1982.
20 M. Größer. *Reduction Meth. for Prob. Model Checking*. PhD thesis, TU Dresden, 2008.
21 J. Kemeny and J. Snell. *Denumerable Markov Chains*. Springer-Verlag, 1976.
22 N.A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.
23 M.L. Puterman. *Markov Decision Processes: Discrete Stocastic Dynamical Programming*. John Wiley & Sons, 1994.
24 J. M. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*. AMS, 2004.
25 M. Vardi. Automatic verification of probabilistic concurrent systems. In *Proceedings of FOCS*, pages 327–338, 1985.

Two Size Measures for Timed Languages

Eugene Asarin[1] and Aldric Degorre[2]

1 **LIAFA – Université Paris Diderot and CNRS, France**
`eugene.asarin@liafa.jussieu.fr`
2 **Université Libre de Bruxelles, Belgium**
`aldric.degorre@ulb.ac.be`

Abstract

Quantitative properties of timed regular languages, such as information content (growth rate, entropy) are explored. The approach suggested by the same authors is extended to languages of timed automata with punctual (equalities) and non-punctual (non-equalities) transition guards. Two size measures for such languages are identified: mean dimension and volumetric entropy. The former is the linear growth rate of the dimension of the language; it is characterized as the spectral radius of a max-plus matrix associated to the automaton. The latter is the exponential growth rate of the volume of the language; it is characterized as the logarithm of the spectral radius of a matrix integral operator on some Banach space associated to the automaton. Relation of the two size measures to classical information-theoretic concepts is explored.

Keywords and phrases timed automata, entropy, mean dimension

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.376

1 Introduction

In a previous work [4, 3], we have formulated the problem of measuring the size (or information content) of a timed regular language. There, we have associated with a language L the volume $\mathcal{V}(L_n)$ of all its words of size n. This volume grows (or vanishes) exponentially as $n \to \infty$, and its rate (i.e. $\lim_n \log \mathcal{V}(L_n)/n$) is referred to as entropy $\mathcal{H}(L)$ of the language. In [4, 3], we characterize this entropy as spectral radius of an integral operator and give some methods to approximately compute it.

The volume-based definition of entropy has, however, some weaknesses when the automaton contains "punctual" transitions guarded by clock constraints of the form $x = c$. Indeed, as soon as a run of the automaton includes such a punctual transition, the volume corresponding to this run becomes 0. Hence, the information content of such a run is disregarded. For example, in the automaton on Fig. 1A, intuitively there are more runs on ba^* than on a^*, however, according to our previous definitions, the volume of all the runs starting by b is 0 (because they should cross a punctual edge), and they are disregarded.

Worse, if all the runs of some automaton include punctual transitions, the entropy becomes $\log 0 = -\infty$ and does not adequately represent the information content.

In this paper, we address the problem of adequately measuring the language size/ information content of a timed language accepted by a timed automaton with punctual and non-punctual transitions. Our solution is freely inspired by some ideas of symbolic dynamics [9], and especially by Gromov's mean dimension, see [10].

The first difficulty is conceptual: the language (up to n events) corresponding to a timed automaton from a geometrical standpoint is a set of polyhedra in $\mathbb{R}^n$. Without punctual transitions, all these polyhedra are full-dimensional, and their cumulated volume is a good aggregate measure of the language. Whenever we allow punctual transitions, the dimension of polyhedra varies from 0 to n, and it becomes more difficult to find their total size. For

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 376–387

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

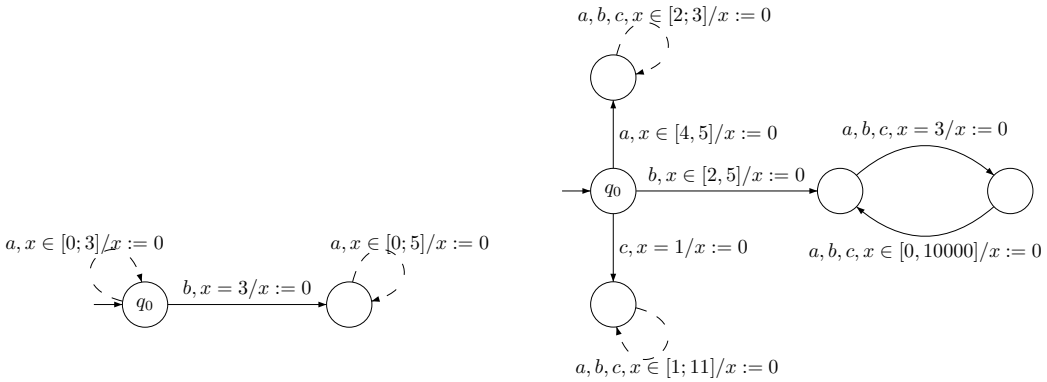


Figure 1 A. Is it reasonable not to go right? B. Three paths. Who will win?

example, in the automaton of Fig. 1B, the geometric set contains some (exponentially many) n-dimensional rectangles of volume 1 corresponding to the words $a\Sigma^*$, some $n-1$-dimensional rectangles of volume 10^{n-1}, and some $(n-1)/2$-dimensional rectangles of volume $3 \cdot 100^{n-1}$. A priori it is not clear how to sum up all these volumes. To address this difficulty, we measure the size of a multidimensional polyhedral set S using a variant of ε-entropy from [7], which corresponds to the amount of information (in bits) needed to specify any point of S with precision ε.

Applying this approach to a regular timed language L, we show that a typical timed word of L_n, whenever time is measured with precision ε, contains $n(\alpha \log(1/\varepsilon) + \mathcal{H})$ bits of information. Thus, the size (growth rate, information production rate) of L is characterized by two numbers $(\alpha, \mathcal{H})$ referred to as *mean dimension and v-entropy*[1]. Roughly, L_n resembles to an αn-dimensional subset of $\Sigma^n \times \mathbb{R}^n$ of a volume $2^{n\,\mathcal{H}}$.

The main result of this paper is a characterization of α and $\mathcal{H}$ of the language L accepted by a timed automaton $\mathcal{A}$ which proceeds as follows. After pre-processing the automaton (splitting its states in regions and removing unreachable states), we obtain a timed automaton $\mathcal{A}'$ and associate with it a max-plus matrix Φ (a kind of adjacency matrix of $\mathcal{A}'$). The mean dimension $\alpha(L)$ is the max-plus spectral radius of Φ. The eigenspace corresponding to this spectral radius leads to identification of several *critical sub-automata* $\mathcal{A}_c$, where all the paths have the same mean dimension α. For each such subautomaton $\mathcal{A}_c$, we build an integral operator Ψ_c acting on a space of functions on the state space of $\mathcal{A}_c$. The v-entropy is the logarithm of the largest (among all the critical components) spectral radius of Ψ_c.

The paper is structured as follows. In Section 2, we introduce and illustrate the notion of ε-entropy and define mean dimension and v-entropy of a timed language. In Section 3, we make some assumptions on timed automata and describe how to preprocess them, and characterize volumes and dimensions of polyhedra in a timed regular language. In Section 4, we obtain the characterization of mean dimension as spectral radius of a natural max-plus matrix Φ (Theorem 9). We also describe the construction of the "critical sub-automata" $\mathcal{A}_c$. In Section 5, we associate to each $\mathcal{A}_c$ a Banach space and a positive linear operator Ψ_c on this space. We characterize v-entropy in terms of its spectral radius in Theorem 14. We also discuss how this spectral radius can be computed in practice. In Section 6, we give an information-theoretic interpretation of α and $\mathcal{H}$ in terms of ε-entropy, which can

[1] "v" for volumetric.

be considered as correctness result for our algorithms. We conclude in Section 7, where we discuss related work and perspectives.

2 Timed languages and their size measures

2.1 Some geometric terminology

A *convex polyhedron* $P \subset \mathbb{R}^d$ is a bounded finite intersection of half-spaces. If it is a subset of some k-dimensional affine subspace, but not of any $k-1$-dimensional one, we say that $\dim P = k$. A *polyhedral set* P is a finite union of convex polyhedra. It can be decomposed into polyhedral components P^m of various dimensions m from 0 to $\dim P$. Such a decomposition could be non-unique, but we will always use a greedy algorithm: find maximal polyhedral subset of maximal dimension - this is the first component. Remove it from P, and repeat the procedure.

The notions above easily extend to subsets of $S \times \mathbb{R}^d$, with S a finite set. Given such a subset P, we denote its component corresponding to an $s \in S$ by P_s, that is $P_s = \{x | (s,x) \in P\} \subset \mathbb{R}^d$. We call a subset P polyhedral, if every component P_s is a polyhedral set. The dimension of a polyhedral set P is the maximum of dimensions of its components. P_s can be further decomposed into subcomponents of different dimensions P_s^m, with $m \leq d$.

In this paper, we will use the well-known ∞-metric on $\mathbb{R}^d$ defined as follows:

$$d(\mathbf{x}, \mathbf{x}') = \max_i |x_i - x'_i|,$$

balls in this metric are cubes. It can be naturally extended to $S \times \mathbb{R}^d$:

$$d((s,\mathbf{x}),(s',\mathbf{x}')) = \begin{cases} d(\mathbf{x},\mathbf{x}'), & \text{if } s = s'; \\ \infty, & \text{otherwise.} \end{cases}$$

2.2 Size of multidimensional sets

The key to measuring such multidimensional sets is provided by Kolmogorov and Tikhomirov's theory of ε-capacity and ε-entropy [7]. We will use a "diametric" variant of the notion of ε-entropy, following [11]. Given a compact metric space X, and a $\varepsilon > 0$, we define the ε-entropy of X as logarithm of the minimum cardinality of a partition of X into (Borel) subsets of a diameter $\leq \varepsilon$. The ε-entropy can be seen as the amount of information (in bits) that is necessary to represent an arbitrary point in X with precision ε.

In particular, for an m-dimensional polyhedron P of a volume V, the minimum cardinality of an ε-partition is close to V/ε^m, thus its logarithm is

$$h_\varepsilon(P) \approx \log V - m \log \varepsilon.$$

For a disjoint union of finitely many polyhedra P_i of dimension m_i and volume V_i, its ε-partition has a size close to $\sum_i V_i/\varepsilon^{m_i}$, (for small ε) and its logarithm can be considered again as information content. Fig. 2 illustrates this simple fact.

This justifies the following:

▶ **Definition 1.** Formal ε-entropy of a finite disjoint family of polyhedra P_i of dimension m_i and volume V_i is defined as:

$$\mathfrak{h}_\varepsilon(P) = \log \sum_i V_i/\varepsilon^{m_i}.$$

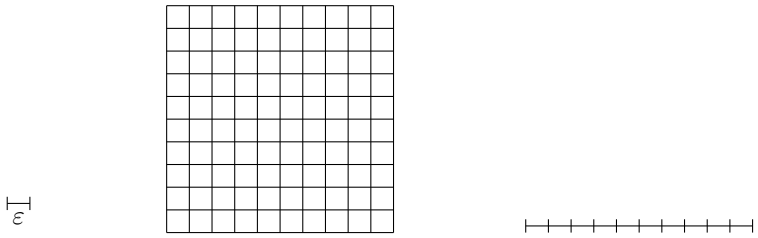


Figure 2 Adding meters to square meters: two polyhedra and their minimal ε-partitions.

2.3 Timed languages and their polyhedra

Given a finite alphabet Σ, a *timed word* is a sequence $t_1a_1t_2a_2\dots t_na_n$ with *events* $a_i \in \Sigma$ and *delays* $t_i \in [0;\infty)$. A *timed language* is just a set of timed words. We will use a natural geometrical interpretation of timed words and languages. Thus a timed word $w = t_1a_1t_2a_2\dots t_na_n$ can be seen as a couple of a discrete word $\eta(w) = a_1a_2\dots a_n$ (called *untiming* of w) and a point $\theta(w) = (t_1, t_2, \dots, t_n) \in \mathbb{R}^n$ called its *timing*, or equivalently as a point in $\Sigma^n \times \mathbb{R}^n$. Similarly, we associate with a timed language L and a natural n, a subset $L_n \subset \Sigma^n \times \mathbb{R}^n$.

For L a timed regular language, all the geometrical sets described above are polyhedral.

In this paper, we will explore dimensionality and volume characteristics of L, such as $\dim(L_n)$, the dimension of the set of n-event timed words in L, and $\mathcal{V}(L_n^m)$, the volume of the m-dimensional component of this set (clearly, it can be non zero only for $m \le \dim(L_n)$).

2.4 Main definitions

The precise aim of this article is to explore the asymptotic behavior of

- $\dim(L_n)$ as $n \to \infty$ (we will show that it is linear);
- $\mathcal{V}(L_n^m)$ as $n \to \infty$ and $m \approx \dim(L_n)$ (we will show that it is exponential).

We will characterize this behavior by two rates (which are just real numbers):

▶ **Definition 2.** Two size measures of a timed regular language are defined as follows:

- *Mean dimension* $\alpha(L) = \lim_{n\to\infty} \dim(L_n)/n$;
- *v-entropy* $\mathcal{H}(L) = \lim_{n\to\infty} \log \max\{\mathcal{V}(L_n^m) \mid \alpha n - d < m < \alpha n + d\}/n$ (with a constant d specified below).

The definition of $\alpha(L)$ is very natural, it says that L_n has dimension $\approx \alpha n$. The definition of $\mathcal{H}(L)$ saying that the volume of L_n^m is approximately $2^{n\mathcal{H}}$ for dimension m close to its maximal possible value, also seems plausible, the only possible doubt is related to the choice of m. We will justify these definitions by Theorem 18, which relates α and $\mathcal{H}$ to the formal ε-entropy $\mathfrak{h}_\varepsilon(L_n)$.

2.5 Example

To illustrate the notions above, consider again the timed language of the automaton of the Fig. 1B. Here we have the choice to explore three different areas of the automaton, depending on the choice of the first symbol between a, b and c, yielding the following language: $L_B = [4;5]a([2;3]\Sigma)^* + [2;5]b(3\Sigma[0;10000]\Sigma)^* + 1c([1;11]\Sigma)^*$. The first branch produces 3^{n-1} copies of the n-dimensional rectangle $[4;5] \times [2;3]^{n-1}$, the second the same number of rectangles $[2;5] \times (\{3\} \times [0;10000])^{(n-1)/2}$ of dimension $(n+1)/2$ (we suppose n

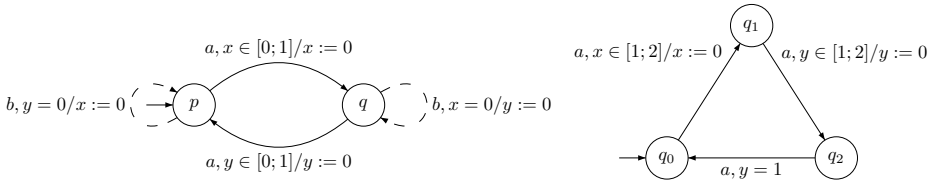


Figure 3 Two interesting automata

odd). The third branch has as many $n-1$-dimensional rectangles $\{1\} \times [1;11]^{n-1}$. Clearly $\dim(L_n) = n$, thanks to the first branch. Hence, the mean dimension $\alpha(L) = 1$. Consider now the volumes:

$$\mathcal{V}(L_n^n) = 3^{n-1}; \quad \mathcal{V}(L_n^{(n+1)/2}) = 300^{n-1} \cdot 3; \quad \mathcal{V}(L_n^{n-1}) = 30^{n-1}.$$

According to Def. 2, v-entropy takes into account only the first and the third volumes (with $m \approx \alpha n = n$, and not the second one). The third volume wins the race, hence the v-entropy $\mathcal{H}(L) = \log 30 \approx 4.9$.

In order to understand why the huge second volume has been disqualified, let us consider the formal ε-entropy of L_n. The sum under logarithm would be:

$$\sum_i V_i/\varepsilon^{m_i} = 3^{n-1}\varepsilon^{-n} + 300^{n-1} \cdot 3\varepsilon^{(1-n)/2} + 30^{n-1}\varepsilon^{1-n}$$
$$= \varepsilon^{-n}\left(3^{n-1} + (300\varepsilon^{1/2})^{n-1} \cdot 3\varepsilon + 30^{n-1}\varepsilon\right).$$

It shows that for $\varepsilon < \frac{1}{100}$, the third term, coming from the sublanguage $1c([1;11]\Sigma)^*$ is the preponderant one, despite the huge interval in $[2;5]b(3\Sigma[0;10000]\Sigma)^*$, which appears only every 2 events. Only terms with dimension m close to the maximum αn can contribute.

In the example considered, we were able to compute mean dimension, v-entropy and formal ε-entropy directly from definitions. To convince the reader that it is not always possible, and advanced methods could be useful, we propose to consider the two automata on Fig. 3. We believe that their size parameters (at least v-entropy) cannot be obtained using elementary methods. We will use the automaton on the right of the figure as a running example.

3 Good timed automata and their pre-processing

We will be able to compute α and $\mathcal{H}$ for a subclass of timed automata, and, before proceeding we have to put the automaton in some normal form. The subclass and the normal form are very close to those from [3], the main difference is that punctual transitions are allowed.

We consider the following variant of Alur and Dill's timed automata (see [1] for original definition). A *timed automaton* (TA) is a tuple $\mathcal{A} = (Q, \Sigma, C, \Delta, q_0)$. Its elements are respectively the set of locations, the alphabet, the set of clocks, the transition relation, and the initial location (we do not need to specify accepting states since all the states are accepting, neither do we need any invariants). A generic state of $\mathcal{A}$ is a pair $(q, \mathbf{x})$ of a control location and a vector of clock values. A generic element of Δ is written as $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q')$ meaning a transition from q to q' with label a, guard $\mathfrak{g}$ and reset $\mathfrak{r}$. We spare the reader the definitions of a run of $\mathcal{A}$ and of acceptance, but we are obliged to fix some notations. Given an automaton $\mathcal{A}$, we write $L(\mathcal{A})$ or just L for its accepted language, $L_p(\mathbf{x})$ for the language

accepted by the runs starting at $(p, \mathbf{x})$; if we want to also specify the last state, we write $L_{pq}(\mathbf{x})$; finally, for the language accepted along some fixed path $\pi = \delta_1 \dots \delta_n$ in $\mathcal{A}$, we write $L_\pi(\mathbf{x})$. This notation will be freely combined with two indices for dimension, thus $L^m_{pn}(\mathbf{x})$ is the m-dimensional component of the set of traces of n-event runs starting at p with clock values $\mathbf{x}$.

We say that a deterministic timed automaton with all accepting states and all guards bounded by a constant M is *good* if the following Assumption holds:

A1. There exists a $D \in \mathbb{N}$ such that on every run segment of D transitions, every clock is reset at least once.

We say that a good TA $\mathcal{A} = (Q, \Sigma, C, \delta, q_0)$ is in a *region-split form* if the following properties hold:

B1. Each location and each transition of $\mathcal{A}$ is visited by some run starting at $(q_0, 0)$.
B2. For every location $q \in Q$, a unique clock region $\mathbf{r}_q$ (called its *entry region*) exists, such that the set of clock values with which q is entered is exactly $\mathbf{r}_q$. For the initial location q_0, its entry region is the singleton $\{0\}$.
B3. The guard $\mathfrak{g}$ of every non-punctual transition $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q') \in \Delta$ is just one clock region.
B4. The guard $\mathfrak{g}$ of every punctual transition $\delta = (q, a, \mathfrak{g}, \mathfrak{r}, q') \in \Delta$ has a form $x_i = c$.

Similarly to [3], it is easy to prove the following:

▶ **Proposition 3.** *Given a good TA $\mathcal{A}$, a region-split TA $\mathcal{A}'$ accepting the same language can be constructed.*

3.1 Recurrent formulas

Given a region-split automaton $\mathcal{A}$, and a path $\pi = \delta_1 \dots \delta_n$ in this automaton, the timed language $L_\pi(\mathbf{x})$ corresponds to one convex polyhedron in $\Sigma^n \times \mathbb{R}^n$, we will denote its projection on $\mathbb{R}^n$ by $P_\pi(\mathbf{x})$. We will compute this polyhedron, its dimension and volume by recurrence on π, starting with an empty path and adding transitions at its beginning one by one.

For the base case (empty path ϵ), we put

$$P_\epsilon(\mathbf{x}) = \{\mathbf{x}\}, \quad \dim P_\epsilon(\mathbf{x}) = 0, \quad \mathcal{V}(P_\epsilon(\mathbf{x})) = 1.$$

Suppose that we know $P = P_\pi(\mathbf{x})$. The recurrent formulas for $P_{\delta\pi}(\mathbf{x})$, its dimension and volume look differently for punctual and non-punctual δ. We summarize them in Table 1 and deduce the following result:

▶ **Proposition 4.** *The dimension and the volume of $P_\pi(\mathbf{x})$ for $\pi = \delta_1 \dots \delta_n$ are as follows:*

$$\dim P_\pi(\mathbf{x}) = \sum_{i=1}^{n} \phi_{\delta_i}; \tag{1}$$

$$\mathcal{V}P_\pi(\mathbf{x}) = (\psi_{\delta_1} \dots \psi_{\delta_n} 1)(\mathbf{x}). \tag{2}$$

Knowing the volume and the dimension for every path, we can in principle compute[2] them

[2] Iterated integrals in the chain of ψ_δ only lead to polynomials and can be easily computed symbolically.

	Punctual δ	Non-punctual δ
δ	$(q, a, y = c, \mathfrak{r}, q')$	$(q, a, \mathfrak{g}, \mathfrak{r}, q')$
$P_{\delta\pi}(\mathbf{x})$	$\{c - y\} \times P_\pi(\mathfrak{r}(\mathbf{x} + c - y))$	$\bigcup_{\tau:\mathbf{x}+\tau\in\mathfrak{g}} \{\tau\} \times P_\pi(\mathfrak{r}(\mathbf{x} + \tau))$
$\dim P_{\delta\pi}(\mathbf{x}) = \dim P_\pi(\mathbf{x}) + \phi_\delta$		
ϕ_δ	0	1
$\mathcal{V}(P_{\delta\pi}(\mathbf{x})) = (\psi_\delta \mathcal{V} P_\pi)(\mathbf{x})$		
$(\psi_\delta v)(\mathbf{x})$	$v(\mathfrak{r}(\mathbf{x} + c - y))$	$\int_{\mathbf{x}+\tau\in\mathfrak{g}} v(\mathfrak{r}(\mathbf{x} + \tau))\, d\tau$

Table 1 Recurrence equations for polyhedra, dimensions, and volumes. ψ_δ is an operator transforming functions to functions.

(for given n and m) for the whole language and its sublanguages, in particular

$$\dim(L_n) = \max\left\{\sum_{i=1}^{n} \phi_{\delta_i} | \delta_1 \dots \delta_n \text{ a path from } (q_0, 0)\right\}; \tag{3}$$

$$\mathcal{V}(L_n^m) = \sum\left\{\psi_{\delta_1} \dots \psi_{\delta_n}(1) | \delta_1 \dots \delta_n \text{ a path from } (q_0, 0) \text{ s.t. } \sum_{i=1}^{n} \phi_{\delta_i} = m\right\}. \tag{4}$$

In two subsequent sections we will determine the asymptotical behavior of these quantities.

4 Max-plus and mean dimension

In this section, we show that $\dim L_n$ is approximately linear wrt n and compute the rate of this dependence. We also clean up the automaton removing the paths that do not give the maximal dimension. The techniques used come from max-plus algebra (see [5]).

4.1 Recalling max-plus

Consider the set $\mathbb{R}_{\max} = \mathbb{R} \cup \{-\infty\}$ endowed with two operations: max ("addition" denoted $\oplus$) and $+$ ("multiplication" denoted $\otimes$). Operations $\oplus$ and $\otimes$ are extended in a natural way to vectors and matrices; A^n denotes the n-th max-plus power of a matrix A.

Similarly to usual linear algebra, for a matrix A we say that $\lambda \in \mathbb{R}_{\max}$ and $\mathbf{x} \in \mathbb{R}_{\max}^n$ are respectively an *eigenvalue* and an *eigenvector* of A if $A \otimes \mathbf{x} = \lambda \otimes \mathbf{x}$. The highest eigenvalue called *spectral radius* of A admits an interpretation in terms of paths in weighted graphs. An $n \times n$ matrix A corresponds to a weighted graph G with vertices $1, \dots, n$. Whenever $A_{ij} > -\infty$, there is an edge (i, j) in the graph, its weight is A_{ij}. For a path π in G, its weight $w(\pi)$ is the sum of weights of edges, and its mean weight is just $w(\pi)/|\pi|$. It is easy to see that A_{ij}^n is the maximal weight of a path of n edges from i to j.

As for the spectral radius α, it can be characterized as the maximum of mean weights for all circuits in the graph G. All the circuits having the same mean weight α are called critical. The critical subgraph $G_{\text{crit}} \subset G$ contains all the vertices and the edges of critical circuits (see an example on Fig. 4). We will need the following well-known results on weights of paths and powers of max-plus matrices:

▶ **Proposition 5** (see [5]). *Let A be a max-plus matrix, $\alpha(A)$ its spectral radius, G_{crit} its critical subgraph. Then,*

- *α and G_{crit} can be computed in $O(n^3)$ using Karp's algorithm;*
- *G_{crit} is a union of several disjoint strongly connected graphs (critical components);*

$$A = \begin{pmatrix} -\infty & -\infty & \mathbf{1} & 1 & -\infty & -\infty \\ \mathbf{2} & \mathbf{1} & -\infty & -\infty & -\infty & 1 \\ -\infty & \mathbf{0} & -\infty & -\infty & -\infty & -\infty \\ -\infty & -\infty & -\infty & -\infty & -\infty & -\infty \\ -\infty & 0 & -\infty & 1 & \mathbf{1} & -\infty \\ -\infty & -\infty & -\infty & -\infty & 0 & -\infty \end{pmatrix}$$

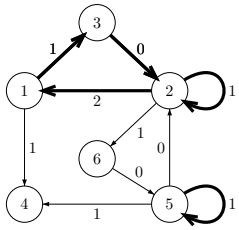


Figure 4 A matrix and a graph; critical edges represented in thick lines, $\alpha = 1$.

- *for i and j in the same critical component, all the n-paths in G_{crit} from i to j have the same weight equal to the maximal weight in the original graph: $= A^n_{ij}$, and close to αn: more precisely for some constant d we have $|A^n_{ij} - \alpha n| < d$;*
- *for arbitrary i and j only the upper bound holds: $A^n_{ij} < \alpha n + d$.*

We see that α is the optimal asymptotic mean weight, and that it is attained by any path in (an SCC of) the critical graph. On the other hand, any path visiting often enough non-critical edges has a lesser weight:

▶ **Proposition 6.** *For any path π in a matrix of $\{-\infty, 0, 1\}^{n\times n}$ with m non-critical and k critical edges, the following upper bound holds:*

$$w(\pi) < d + \alpha k + \beta m,$$

with some constant $\beta < \alpha$ which depends only on the matrix A.

4.2 Matrix Φ

The theory described above applies almost directly to dimension of timed polyhedra. Indeed, let us define a max-plus matrix Φ such that $\Phi_{pq} = \max\{\phi_\delta \mid \delta \text{ from } p \text{ to } q\}$. In other words,

$$\Phi_{pq} = \begin{cases} -\infty & \text{if there is no transition from } p \text{ to } q; \\ 0 & \text{if there is a transition and all transitions are punctual;} \\ 1 & \text{if there is a non-punctual transition.} \end{cases}$$

Then, the following is almost immediate from (3).

▶ **Proposition 7.** *Φ^n_{pq} is the dimension of $L_{pqn}(\mathbf{x})$ (for any clock valuation $\mathbf{x}$).*

Thus, we can apply the general theory, compute the spectral radius $\alpha(\Phi)$, the critical subgraph and its SCC decomposition. This way, for any SCC c we obtain a *critical subautomaton* $\mathcal{A}_c$ whose control states are the ones corresponding to vertices of c and whose transitions are those of $\mathcal{A}$, going along critical edges, and having maximal dimension for each such edge[3]. Applying Prop. 5 to $\mathcal{A}_c$ we obtain:

▶ **Corollary 8.** *In each critical subautomaton $\mathcal{A}_c$, for any n-path π from p to q , the dimension of $P_\pi(\mathbf{x})$ does not depend on the choice of the path. It is close to αn: more precisely for some constant d we have $|\dim P_\pi(\mathbf{x}) - \alpha n| < d$.*

And we deduce the main result of this section:

[3] We do not define the initial state for $\mathcal{A}_c$.

▶ **Theorem 9.** *For any good region-split automaton $\mathcal{A}$, let $\alpha > -\infty$ be the spectral radius of its matrix Φ. Then, for some constant d and all n, the language L of $\mathcal{A}$ satisfies:*

$$|\dim L_n - \alpha n| < d.$$

Thus α is the mean dimension of L. (In the degenerate case when $\alpha = -\infty$, the automaton is acyclic and L_n is empty for n large enough.)

▶ **Example 10.** We consider the automaton on the right of Fig. 3 and put it in region-split form. We call p_0, p_1 and p_2 the region-split states corresponding to q_0, q_1 and q_2 with respective entry regions $[y = 1 < x < 2]$, $[0 = x < 1 < y]$ and $[y = 1 < x < 2]$. The only critical cycle is then $p_0 \xrightarrow{1<x<2,\ x:=0} p_1 \xrightarrow{1<y<2,\ y:=0} p_2 \xrightarrow{y=1} p_0$. As it yields two non-punctual and one punctual transition, its mean dimension, the max-plus spectral radius of Φ, is $\frac{2}{3}$. Therefore, in this example, $\alpha = \frac{2}{3}$.

5 Functional analysis and v-entropy

Exploration of the asymptotic behavior of the volume goes along the same lines as for dimension, but instead of a max-plus matrix, a matrix of integral operators is iterated. As a result, while dimension's asymptotics is linear, the volume evolves exponentially.

5.1 Recalling functional analysis

In order to characterize and compute the v-entropy $\mathcal{H}$, we will use the approach introduced in [3], based on functional analysis (see e.g. [12]) and in particular the theory[4] of positive linear operators on Banach spaces (see [8]). We will need the following result:

▶ **Theorem 11** (see [8]). *Given a positive linear bounded compact operator Θ with a spectral radius $\rho > 0$, defined on a Banach space ordered by a generating cone [5], the following holds:*

1. *ρ is an eigenvalue;*
2. *there exists a non-negative eigenvector $f \geq 0$ corresponding to this eigenvalue;*
3. *Gelfand's formula holds: $\lim_n ||\Theta^n||^{1/n} = \rho$.*

5.2 Banach space and operator Ψ

Similarly to [3], and to the previous section, we will represent equation (4) as iteration of some positive operator, and apply Theorem 11. However, we must take into account the two parameters n and m of the volume $\mathcal{V}(L_n^m)$. Our solution is as follows: we will consider the critical subgraph introduced above, and thus concentrate on the polyhedra of maximal dimension $m \approx \alpha n$ (which corresponds to the definition of v-entropy).

Given a good timed automaton in the region-split form, we first restrict to one critical subautomaton $\mathcal{A}_c$ (we denote its state space by Q_c). We define the Banach space $\mathcal{F}_c$ as the set of continuous bounded functions on the set $\{(q, \mathbf{x}) | q \in Q_c, \mathbf{x} \in \mathbf{r}_q\}$. The norm is defined by $||f|| = \sup_{q,\mathbf{x}} |f(q, \mathbf{x})|$. An element $f \in \mathcal{F}$ can be seen as a vector of functions f_q with simplicial domains $\mathbf{r}_q$.

[4] A generalisation of the classical Perron-Frobenius theory to infinite dimension.
[5] The space $\mathcal{F}$ of continuous function considered below straightforwardly satisfies these properties.

For any $p, q \in Q_c$, we define an operator ψ_{pq} which maps functions over $\mathbf{r}_q$ to functions over $\mathbf{r}_p$ as the sum of all the operators ψ_δ (defined in Table 1) for the transitions going from p to q. Next we define an operator Ψ_c on $\mathcal{F}$ with the matrix $(\psi)_{pq}$:

$$(\Psi_c f)(p, \mathbf{x}) = \sum_q (\psi_{pq} f_q)(\mathbf{x}).$$

The operator Ψ_c provides a simple form to formulas (4) restricted to critical paths and maximal dimension. Let L^c (with subscripts and arguments interpreted in the standard way) denote the language of the critical subautomaton $\mathcal{A}_c$.

▶ **Proposition 12.** *For any $p, q \in Q_c$:*

$$\mathcal{V}(L^c_{pqn}(\mathbf{x})) = (\Psi_c^n 1_q)_p(x),$$

where the function 1_q equals one on the state q (more precisely, on $\{q\} \times \mathbf{r}_q$), and zero elsewhere.

5.3 Characterization of v-entropy

In Prop. 12, we have characterized the volume of $L^c_{pqn}(\mathbf{x})$ in terms of nth iteration of the matrix integral operator Ψ_c. On the other hand, this operator almost satisfies the hypotheses of Theorem 11:

▶ **Proposition 13.** *Ψ_c is a bounded linear positive operator. Ψ_c^{D+1} is compact with D as in Assumption A1.*

This allows to prove, using Theorem 11:

▶ **Theorem 14.** *Let $\mathcal{A}$ be a good region-split TA, $\mathcal{A}_c$ its critical subautomaton, Ψ_c the operator described above for $\mathcal{A}_c$, and ρ_c the spectral radius of this operator. Then the following holds:*

- *for any $\sigma > 0$, and n big enough, for any state $(q, \mathbf{x})$ of $\mathcal{A}_c$, the upper bound:*
$$\mathcal{V}(L^c_{qn}(\mathbf{x})) < (\rho_c + \sigma)^n;$$
- *and for some state q of $\mathcal{A}_c$, some open set O inside $\mathbf{r}_q$ and some $\gamma > 0$, for all $\mathbf{x} \in O$, for all n, the lower bound:*
$$\mathcal{V}(L^c_{qn}(\mathbf{x})) > \gamma \rho_c^n.$$

The theorem says that in a critical $\mathcal{A}_c$, the volume grows roughly as ρ_c^n, or, in exponential form, as $2^{n \log \rho_c}$. We can deduce now the required characterization of the v-entropy of the whole language of $\mathcal{A}$.

▶ **Theorem 15.** *Let $\mathcal{A}$ be a good region-split TA. Then its v-entropy $\mathcal{H}$ is the maximum of $\log \rho_c$ over all its critical subautomata $\mathcal{A}_c$.*

▶ **Example 16.** On the only critical component of the automaton on the right of Fig. 3 (containing only the three transitions of the critical cycle we mentioned earlier in Ex. 10), the operators ψ_{pq} (defining Ψ_c) have the following expressions:

$$(\psi_{p_0 p_1} f)(x, y) = \int_{\tau=0}^{2-x} f(0, y + \tau) d\tau;$$
$$(\psi_{p_1 p_2} f)(x, y) = \int_{\tau=0}^{2-y} f(x + \tau, 0) d\tau;$$
$$(\psi_{p_2 p_0} f)(x, y) = f(x + 1, 1).$$

A close examination shows that the integral system $\Psi_c f = \lambda f$ can be rewritten using only one real variable, then differentiated twice and finally solved symbolically as a linear ordinary differential equation. Doing so, we find that the λ having the highest absolute value such that the system still has non-trivial solutions is $\left(\frac{2}{\pi}\right)^{2/3}$. Therefore $\mathcal{H} = \frac{2}{3}\log\frac{2}{\pi}$.

5.4 Algorithmic aspects

Practical computation of the spectral radius of an operator Ψ represented by a matrix of integral operators is a nontrivial task. However, the two methods proposed in [3] can be applied almost without change. We refer the reader to [3, 2], and only sketch the two methods:

- The first one applies to the subclass of "1 1/2 clocks" automata such that all the regions $\mathbf{r}_q$ are of dimension 0 or 1 (it means that all the clocks but one should be reset when taking a transition). For such an automaton it is possible to transform the integral eigenvalue equation $\Psi v = \lambda v$ to a system of linear ordinary differential equations (indeed, unknown functions v_q are functions of scalar arguments), solve it symbolically and thus obtain a closed-form equation on the largest eigenvalue ρ.
- The second (numerical) method uses iterations of operator Ψ. It is based on the following fact on positive operators from [8]:
  ▶ **Proposition 17.** *Let* $v_n = \Psi^n 1$, *and* $\alpha = \min_{q,x} \frac{v_{n+1}(q,x)}{v_n(q,x)}$; $\beta = \max_{q,x} \frac{v_{n+1}(q,x)}{v_n(q,x)}$. *Then* $\alpha \le \rho(\Psi) \le \beta$.
  The bounds α and β can be obtained by a straightforward symbolic computation.

6 Size versus information

▶ **Theorem 18.** *Let L be a timed language of a good automaton, α and $\mathcal{H}$ its mean dimension and v-entropy. If $\alpha > 0$ and $\mathcal{H} > -\infty$, then the formal ε-entropy of L_n satisfies the inequality, for all $\eta > 0$, for ε small enough and n large enough:*

$$n(-\alpha \log \varepsilon + \mathcal{H} - \eta) \le \mathfrak{h}_\varepsilon(L_n) \le n(-\alpha \log \varepsilon + \mathcal{H} + \eta).$$

The second term of this inequality is logarithm of $\sum_m \mathcal{V}(L_n^m)\varepsilon^{-m}$ where the sum is computed over all the dimensions m. The first and third terms are close to the logarithm of the similar sum computed only for terms with $m \approx \alpha n$. The proof of the upper bound in Theorem 18 is based on technical estimates showing that contribution of terms with $m < \alpha n$ in the sum can be reasonably upper bounded.

7 Conclusions

This paper reports progress achieved recently in the information-theoretical studies of timed regular languages, especially with punctual transitions. Two components of information content have been identified and characterized: one represents the dimensionality, another the volume. Both are characterized by spectral radii of linear operators, matrices of both operators reproduce the structure of the automaton, but they are still very different in nature (one is max-plus and finite-dimensional, other is a classical integral operator). Our understanding of the role of punctual transitions in timed languages has substantially improved. As ongoing and future work we are interested in relating these results to other information measures, such as Kolmogorov complexity, and topological entropy, in the spirit of symbolic dynamics. Our work went in parallel and in interaction with the MSc thesis [6] supervised by one

of us and exploring in depth symbolic dynamics of timed automata. These two research lines would eventually merge. On the other hand, we believe that improved measures of information content for a larger class of timed languages introduced here are more suitable for eventual implementation and applications.

Acknowledgments

We are thankful to Dominique Perrin for encouraging discussions, to Mike Boyle for attiring our attention to mean dimension, and to Nicolas Basset for sharing his inspiring ideas on timed symbolic dynamics.

References

1 R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

2 E. Asarin and A. Degorre. Volume and entropy of regular timed languages. Preprint, 2009. http://hal.archives-ouvertes.fr/hal-00369812/.

3 E. Asarin and A. Degorre. Volume and entropy of regular timed languages: Analytic approach. In *FORMATS'09*, LNCS 5813, pages 13–27, 2009.

4 E. Asarin and A. Degorre. Volume and entropy of regular timed languages: Discretization approach. In *Concur'09*, LNCS 5710, pages 69–83, 2009.

5 F. Baccelli, G. Cohen, G. J. Olsder, and J.-P. Quadrat. *Synchronization and Linearity*. 2001. http://www-rocq.inria.fr/metalau/cohen/documents/BCOQ-book.pdf.

6 N. Basset. Dynamique symbolique et langages temporisés. Master's thesis, Master Parisien de la Recherche Informatique, 2010.

7 A. Kolmogorov and V. Tikhomirov. ε-entropy and ε-capacity of sets in function spaces. *Uspekhi Mat. Nauk*, 14(2):3–86, 1959. Russian, partial English translation in [**?**].

8 M. Krasnosel'skij, E. Lifshits, and A. Sobolev. *Positive Linear Systems: the Method of Positive Operators*. Heldermann Verlag, Berlin, 1989.

9 D. Lind and B. Marcus. *An Introduction to Symbolic Dynamics and Coding*. Cambridge University Press, 1995.

10 E. Lindenstrauss and B. Weiss. Mean topological dimension. *Israel J. of Math.*, 115:1–24, 2000.

11 E. C. Posner and E. R. Rodemich. Epsilon entropy and data compression. *Ann. Math. Statist*, 42(6):2079–2125, 1971.

12 B. Rynne and M. Youngson. *Linear Functional Analysis*. Springer, 2008.

Average Analysis of Glushkov Automata under a BST-Like Model

Cyril Nicaud[1], Carine Pivoteau[1], and Benoît Razet[2]

1 **LIGM, Univ. Paris-Est, CNRS UMR 8049, France**
{cyril.nicaud,carine.pivoteau}@univ-mlv.fr

2 **Tata Institute of Fundamental Research, Mumbai, India**
benoit.razet@gmail.com

Abstract

We study the average number of transitions in Glushkov automata built from random regular expressions. This statistic highly depends on the probabilistic distribution set on the expressions. A recent work shows that, under the uniform distribution, regular expressions lead to automata with a linear number of transitions. However, uniform regular expressions are not necessarily a satisfying model. Therefore, we rather focus on an other model, inspired from random binary search trees (BST), which is widely used, in particular for testing. We establish that, in this case, the average number of transitions becomes quadratic according to the size of the regular expression.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.388

1 Introduction

Finite state automata are an essential data structure in computer science, which have been extensively studied since the fifties. Kleene, in his seminal paper [14], introduced regular expressions to describe the behavior of automata and showed a fundamental result: automata and regular expressions define the same objects, regular languages. Regular expressions are widely used in string searching programs and scripting languages such as *grep*, *sed*, *awk*, *Perl*, *Python* and *Ruby*. And most often, programs involving regular expressions are more efficient when the expressions are compiled into automata instead of interpreting them on the fly. The study of algorithms compiling (or one can say translating) regular expressions into automata is therefore a prolific area, which is still very active, in particular because of the large variety of automata and compilation techniques. A classical method to compare the resulting algorithms, besides time and space complexities, is to study the size of the output automaton, defined either as its number of states or of transitions.

In this article we study the average number of transitions of the automaton computed by a famous algorithm proposed by Glushkov [12] and independently by McNaughton and Yamada [16]. The automaton produced is now called Glushkov automaton or position automaton. The position automaton refers to the work of Berry and Sethi [3], who have provided a fast algorithm for compilation that associates to each *position* symbol of an expression, a state in the resulting automaton. This algorithm is also described in the standard textbook on compilers by Aho, Sethi and Ullman [1]. The worst-case complexity analysis on Berry-Sethi's algorithm shows that it produces an automaton with at most a quadratic number of transitions with respect to the size of the input regular expression (its number of symbols). But one may wonder what is the behavior of the algorithm in practice, which naturally leads to consider its average complexity.

IARCS Annual International Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 388–399

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

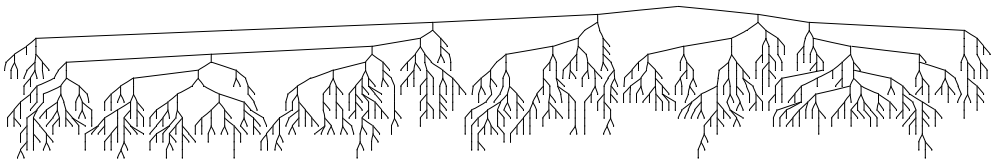


Figure 1 Random unary-binary tree (1000 nodes) according to the BST-like distribution.

A recent work [17] has shown that, considering the uniform distribution on regular expressions, the average number of transitions of Glushkov automaton is in fact linear, when the worst case is quadratic. Since a distribution which is uniform seems to be *a priori* natural, one may conclude that, in practice, the average number of transitions should be linear and not quadratic. Nevertheless, one can argue that this model may not be that relevant. For instance, the number of nested stars in a typical random expression is in $\Theta(\sqrt{n})$, which is much larger than expected. For testing purposes, an other natural distribution appears, inspired from random binary search trees (for short, we call it the *BST-like distribution*). In particular, this distribution has been used to generate random formulas of *Linear Temporal Logic* in order to validate algorithms in [6, 19]. To highlight the difference with the uniform model, Figures 1 and 2 display two random trees according to the two distributions: one shall be struck by their contrasting profiles, in particular looking at their height.

The main result of this paper is that the average number of transitions of Glushkov automaton is quadratic with respect to the size of the regular expression under a BST-like distribution. In this process, we also analyze in details the probability that a random regular expression recognizes the empty word.

This article is organized as follows. In Section 2, we define the BST-like distribution on regular expressions and recall some basic facts on Glushkov automata. The main theorem is given in Section 3. Intermediate results and their proofs are presented in Section 4. Finally, in the concluding section, we give experimental data to illustrate these results. Due to a lack of space, most of the proofs are sketched or omitted in this extended abstract.

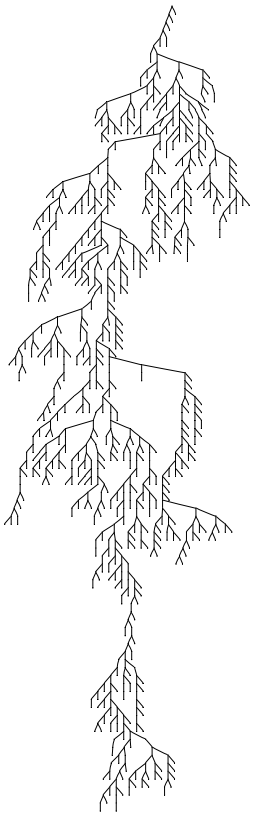


Figure 2 Uniform random unary-binary tree (1021 nodes).

2 Definitions

2.1 The BST-like distribution

We devote this section to the presentation of the trees corresponding to regular expressions, focusing on the fact that a BST-like distribution on such trees is not uniform. Recall that the uniform distribution on a finite set S is achieved by giving the same probability $1/|S|$ to all the elements of S.

2.1.0.1 Unary-binary plane trees

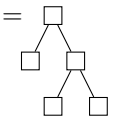
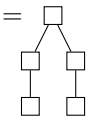
We first consider the classical model of unary-binary trees that are defined inductively as either single nodes (leaves) or nodes having exactly one child or two ordered children that are themselves unary-binary trees. The regular expressions considered in the sequel are a specialization of these trees. The number of nodes of a tree T is called its *size*, denoted by $|T|$. Following the recursive definition, one has a quite natural and simple algorithm `UB(`n`)` to produce a unary-binary tree of size n:

```
UB(n) ------------------------------------------------------------
     if n=1 then return a node (denoted by □)
     if n=2 then return □
                        |
                        □
     else, choose if the root is unary or binary
            if unary then return □
                                |
                                UB(n−1)
            else choose k between 1 and n − 2 and return □
                                                       /\
                                                      /  \
                                                UB(k)    UB(n−k−1)
-----------------------------------------------------------------
```

To transform this procedure into a random sampler, the (unspecified) choices are randomized in order to obey probabilistic laws that dictate the random distribution on the whole set of trees of size n. In this study, we consider the case where a unary node is chosen with probability $q \in]0,1[$ and a binary node with probability $1-q$. The size of the left child of a binary node is drawn uniformly at random. All the choices are independent, thus, in this model, the probability p of a tree is defined inductively by:

$$\begin{cases} p\left(\square\right) & = 1, \\ p\left(\begin{array}{c}\square\\|\\T\end{array}\right) & = q \cdot p(T), \\ p\left(\begin{array}{c}\square\\/\backslash\\T_1\ T_2\end{array}\right) & = \frac{1-q}{n-2} p(T_1)p(T_2), \quad \text{if } |T_1|+|T_2|+1 = n. \end{cases} \tag{1}$$

For any $n \geq 1$, p is a discrete probability measure on the set of unary-binary trees of size n, but one can notice that, for any value of q in $]0,1[$, the probability distribution induced by this definition is not uniform. This is readily checked, observing that the probabilities of the two following unary-binary trees of size 5 cannot match: the equation $p(T_1) = p(T_2)$ has no solution for q in $]0,1[$ when $p(T_1) = (1-q)^2/3$ and $p(T_2) = (1-q)/3$.

$T_1 =$  $T_2 =$ 

Actually, this probabilistic model is a natural extension of what is obtained for binary trees by choosing recursively the sizes of the two subtrees of a node uniformly at random; this corresponds exactly to the common random distribution on binary search trees (see [15]): to build a random BST of size n, nodes are inserted one at a time (using the standard insertion procedure for BST [5]), according to a uniform random permutation of $\{1,\ldots,n\}$. Therefore, from now on, we call this model the *BST-like distribution.*

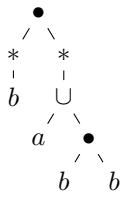


Figure 3 Unary-binary tree of size 9 representing the regular expression $b^* \bullet (a \cup b \bullet b)^*$.

2.1.0.2 Height of a random tree

To emphasize this dissimilarity, one can remark that the asymptotic profiles of the trees according to these two distributions highly differ. This is observable on different parameters, one of the most commonly studied being the height. According to [10], the average height of a large uniform unary-binary tree with n nodes is in $\Theta(\sqrt{n})$, when we show here that it is in $\Theta(\log n)$, according to our distribution. The later result was expectable, since this corresponds to the average height of binary search trees [18, 7, 8]. This explains the difference between the shapes of the two unary-binary trees of Figures 1 and 2, even though they are about the same size.

▶ **Proposition 1.** The average height of a unary-binary tree of size n according to the BST-like distribution is in $\Theta(\log n)$.

Proof. (Sketch) This proof is adapted from the second edition of Introduction to Algorithms [5, p. 265–268]. Let X_n be the random variable associated to the height of the unary-binary trees of size n and let $Y_n = \lambda^{X_n}$, with $\lambda = \frac{3}{2+q}$. Using the fact that the height of a tree that is not reduced to a single node is bounded from above by the sum of the heights of its children plus one, we get that for all positive integer n, $\mathbb{E}[Y_n] \leq y_n$, where $(y_n)_{n\in\mathbb{N}^*}$ is defined by: $y_1 = 1$, $y_2 = \lambda$, and for all $n \geq 3$,

$$y_n = \lambda q y_{n-1} + \frac{2\lambda(1-q)}{n-2} \sum_{\ell=1}^{n-2} y_\ell.$$

Since $\lambda = \frac{3}{2+q}$, one can prove by direct induction that $y_n \leq \binom{n+1}{2}$, for any positive integer n. We conclude by Jensen's inequality, since $\lambda^{\mathbb{E}[X_n]} \leq \mathbb{E}[\lambda^{X_n}] \leq y_n$. ◀

2.2 Random regular expressions

We consider non-empty regular expressions on an alphabet A, represented as unary-binary plane trees. The internal nodes are either the unary star operator $*$ or one of the two binary operators: union $\cup$ and concatenation $\bullet$. The leaves (external nodes) are either letters of A or the empty word ε (see example of Figure 3). Let $\mathcal{T}_n$ denote the set of all regular expressions with n nodes (both internal and external), and $\mathcal{T} = \cup_{n\in\mathbb{N}}\mathcal{T}_n$ be the set of all regular expressions. The size of an expression $T \in \mathcal{T}$ corresponds to the number of nodes of the tree, that is the number of symbols in the expression (excluding parentheses). A language defined on A is *denoted* by a regular expression when it is exactly the set of words obtained by interpreting each symbol $*$, $\bullet$ or $\cup$ as the associated regular operation on sets of words. Let $L(T)$ be the language denoted by $T \in \mathcal{T}$.

We extend the probabilistic model of the unary-binary trees defined in Section 2.1.0.1 to regular expressions as follows. Let $p_\varepsilon \in]0, 1[$ be the probability associated to ε and p

be a mapping from A to $]0,1[$ such that $\sum_{a\in A} p(a) = 1 - p_\varepsilon$. The mapping p is extended inductively to regular expressions by:

$$\begin{cases} p\left(\overset{*}{|}_{T}\right) &= p(T) & \text{if } |T| = 1, \\ p\left(\overset{*}{|}_{T}\right) &= q \cdot p(T) & \text{if } |T| \geq 2, \\ p\left(\overset{\cup}{/\!\backslash}_{T_1\ T_2}\right) &= \frac{1-q}{2(n-2)} p(T_1)p(T_2) & \text{if } |T_1| + |T_2| + 1 = n, \\ p\left(\overset{\bullet}{/\!\backslash}_{T_1\ T_2}\right) &= \frac{1-q}{2(n-2)} p(T_1)p(T_2) & \text{if } |T_1| + |T_2| + 1 = n, \end{cases} \tag{2}$$

where q in $]0,1[$ is the probability for an internal node to be the star operator. One can check by induction on $n \geq 1$ that p is a discrete probability measure on $\mathcal{T}_n$, i.e.,

$$\sum_{T\in\mathcal{T}_n} p(T) = 1. \tag{3}$$

Note that the $\cup$-nodes and the $\bullet$-nodes have the same probability to be generated in this distribution (mostly to keep the following computations trackable).

According to this model, the algorithm `UB(`n`)` producing unary-binary trees transforms into a random sampler `RE(`n`)` for regular expressions. This sampler has been used to generate the random tree displayed by Figure 1, forgetting the labels, with $q = 1/3$. As for the uniform tree of Figure 2, it has been produced by a Boltzmann sampler [9].

```
RE(n) ------------------------------------------------------------
    if n=1 then return ε with proba p_ε or a letter ℓ with proba p(ℓ)
    if n=2 then return (RE(1))*
    else, choose "unary" with proba q or "binary" with proba 1 − q
           if "unary" then return (RE(n − 1))*
           else choose k uniformly at random between 1 and n − 2
                  return RE(k) ∪ RE(n − k − 1) with proba 1/2
                  or return RE(k) • RE(n − k − 1) with proba 1/2
-------------------------------------------------------------------
```

Note that to choose a random element in a set $S = \{s_1, \ldots, s_n\}$, each with probability $p(s_i)$, one simply needs to pick a random value in the interval $I = [0,1[$ and return the element corresponding to the subinterval of I where it belongs, when I is divided according to p:

$$I = [0, p(s_1)[\cup [p(s_1), p(s_1) + p(s_2)[\cup \cdots \cup [1 - p(s_n), 1[.$$

2.3 Glushkov automaton

We give here the formal construction to compute the Glushkov automaton [12, 16, 3] of any regular expression and introduce the notations used in the sequel.

Let m be the number of letter symbols in T, for $T \in \mathcal{T}$. We consider the expression $\widetilde{T}$ obtained from T by distinguishing the letters with subscripts in $\{1, \ldots, m\}$, marking them from left to right on its string representation, or equivalently using depth-first order on its tree representation. For instance $T_0 = b^* \bullet (a \cup b \bullet b)^*$ is changed into $\widetilde{T_0} = b_1^* \bullet (a_2 \cup b_3 \bullet b_4)^*$. We denote by $\mathtt{Pos}(T)$ the set of subscripted letters in $\widetilde{T}$: $\mathtt{Pos}(T_0) = \{b_1, a_2, b_3, b_4\}$ in the example. We also denote by ν the function from $\mathtt{Pos}(T)$ to A that removes the subscripts; for instance, $\nu(a_2) = a$.

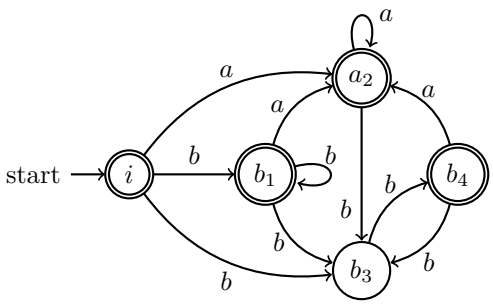


Figure 4 Glushkov automaton for the expression $\widetilde{T_0} = b_1^* \bullet (a_2 \cup b_3 \bullet b_4)^*$.

The automaton construction we study relies on the relative positions of letters in the words recognized by this automaton; thus we introduce the three following sets of distinguished letters that are used to describe these positions. For any regular expression T, let $\texttt{First}(T)$ and $\texttt{Last}(T)$ be the sets defined by

$$\texttt{First}(T) = \{\alpha \in \texttt{Pos}(T) \mid \exists u \in L(\widetilde{T}),\ u \text{ starts with the letter } \alpha\},$$
$$\text{and} \quad \texttt{Last}(T) = \{\alpha \in \texttt{Pos}(T) \mid \exists u \in L(\widetilde{T}),\ u \text{ ends with the letter } \alpha\}.$$

For instance, $\texttt{First}(T_0) = \{b_1, a_2, b_3\}$ and $\texttt{Last}(T_0) = \{b_1, a_2, b_4\}$. And for any letter α in $\texttt{Pos}(T)$, the set $\texttt{Follow}(T, \alpha)$ is defined by

$$\texttt{Follow}(T, \alpha) = \{\beta \in \texttt{Pos}(T) \mid \exists u \in L(\widetilde{T}),\ \alpha \text{ and } \beta \text{ are consecutive letters in } u\}.$$

The *Glushkov automaton* of T, also called the *position automaton*, is the automaton $\mathcal{A}_T$ defined by $\mathcal{A}_T = (A, Q, R, \{i\}, F)$ with $Q = \texttt{Pos}(T) \cup \{i\}$, $F = \texttt{Last}(T) \cup \{i\}$ if $\varepsilon \in L(T)$ and $F = \texttt{Last}(T)$ otherwise, and

$$R = \{i \xrightarrow{\nu(\alpha)} \alpha \mid \alpha \in \texttt{First}(T)\} \cup \{\alpha \xrightarrow{\nu(\beta)} \beta \mid \beta \in \texttt{Follow}(T, \alpha)\}.$$

This classical construction provides an automaton that recognizes $L(T)$. As an example, the Glushkov automaton of T_0 is depicted by Figure 4.

3 Main result

▶ **Theorem 2.** *In the BST-like model, the average number of transitions in the Glushkov automaton of a size n regular expression is quadratic, i.e., in $\Theta(n^2)$.*

Proof. First, assume that the expected size f_n of `First` (its cardinality) is linear with respect to the size n of the regular expression, i.e., that f_n satisfies the asymptotic equivalent $f_n \sim Kn$, for some positive real K that only depends on p_ε and q. The proof of this result (Theorem 7), which is technical, is given in the next section.

Recall that Markov inequality states that if X is a non-negative random variable with expectation $\mathbb{E}[X]$, then for any positive real number a,

$$\mathbb{P}(X \geq a) \leq \frac{\mathbb{E}[X]}{a}.$$

For $n \geq 1$, let $X_n : \mathcal{T}_n \to \mathbb{R}$ be the random variable that associates $n - |\texttt{First}(T)|$ to any $T \in \mathcal{T}_n$. This random variable is non-negative, since $|\texttt{First}(T)|$ is at most n for any

element of $\mathcal{T}_n$. Therefore, setting $a = \alpha n$ in Markov inequality, with $1 - K < \alpha < 1$, we obtain that

$$\mathbb{P}\left(X_n \geq \alpha n\right) \leq \frac{\mathbb{E}[X_n]}{\alpha n},$$

and thus

$$\mathbb{P}\left(|\texttt{First}\left(T\right)| \leq (1-\alpha)n\right) \leq \frac{n - f_n}{\alpha n}.$$

The right quantity is asymptotically equivalent to $\frac{1-K}{\alpha} < 1$, then there exists two real numbers β and γ in $]0,1[$, with $\beta < (1-\alpha)$ and $0 < \gamma < 1 - \frac{1-K}{\alpha}$, such that for n large enough,

$$\mathbb{P}\left(|\texttt{First}\left(T\right)| \geq \beta n\right) \geq \gamma.$$

By symmetry, this result also holds for $\texttt{Last}\left(T\right)$. Moreover, the probability that a regular expression T of size $n+2$ satisfies the following conditions:

$$T = \overset{\bullet}{\underset{T_1}{\wedge}\underset{T_2}{\vee}}, \qquad |T_1| \in \left[\lfloor \tfrac{n}{3} \rfloor, \lceil \tfrac{2n}{3} \rceil\right], \qquad |\texttt{Last}\left(T_1\right)| \geq \beta |T_1| \quad \text{and} \quad |\texttt{First}\left(T_2\right)| \geq \beta |T_2|,$$

is at least, for n large enough,

$$\underbrace{\frac{1-q}{2}}_{\text{root}=\bullet}\ \underbrace{\frac{1}{3}}_{|T_1|}\ \underbrace{\gamma}_{|\texttt{Last}(T_1)|}\ \underbrace{\gamma}_{|\texttt{First}(T_2)|} = \frac{(1-q)\gamma^2}{6} > 0.$$

Note that for any a in $\texttt{Last}\left(T_1\right)$ and any b in $\texttt{First}\left(T_2\right)$, the transition $a \xrightarrow{\nu(b)} b$ is in the automaton, since the letter b is in $\texttt{Follow}\left(T_1 \bullet T_2, a\right)$. Therefore, any tree satisfying the above conditions yields to an automaton with at least $|\texttt{Last}\left(T_1\right)| \cdot |\texttt{First}\left(T_2\right)| \geq \beta^2 n^2$ transitions. Therefore, the expected number of transitions is bounded below by

$$\frac{(1-q)\gamma^2}{6}\beta^2 n^2 = \Omega\left((n+2)^2\right),$$

in the Glushkov automaton of a size $n+2$ expression. The $\mathcal{O}(n^2)$ bound being obvious, the result follows. ◀

The next section is devoted to the exposition of some intermediate results to complete this proof. Among them, the key point is given by Theorem 7, which states that the average size of `First` (resp. `Last`) is linear with respect to the size of the regular expression; considering only the sub-expressions of the form $T_1 \bullet T_2$, one can observe that the number of new transitions they imply in the automata is $|\texttt{Last}\left(T_1\right)| \cdot |\texttt{First}\left(T_2\right)|$, which justify the quadratic number of transitions in the whole automata. The other point is that the size of `First` (resp. `Last`) is highly related to the probability of recognizing the empty word, given by Theorem 3 which states that a large expression recognizes ε with high probability.

4 Some properties of random expressions in the BST-like model

4.1 Analytic tools

In the sequel, the proofs mostly rely on techniques of analytic combinatorics. To study the asymptotic behavior of a sequence $(a_n)_{n\in\mathbb{N}}$, the idea is to consider its *generating function* $A(z)$, which is the formal power series defined by

$$A(z) = \sum_{n\in\mathbb{N}} a_n z^n.$$

From a recursive specification of $(a_n)_{n\in\mathbb{N}}$, one can often get a functional equation satisfied by $A(z)$. At this point, several theorems exist to compute asymptotic estimates of its Taylor coefficients, which are exactly the a_n's. These theorems mainly use the theory of complex analysis, seeing generating functions as analytic functions from $\mathbb{C}$ to $\mathbb{C}$. The main idea is that asymptotic equivalents for the coefficients of a generating function can be obtained by studying it around its dominant singularities (its singularities of smallest moduli).

In this article, the recursive descriptions of sequences lead to ordinary differential equations for their generating functions. These equations can be solved using the well-known variation-of-constants method, and the solutions have similar properties: they have a unique dominant singularity at 1 and satisfy the required analytic conditions. Therefore, provided the expansion of $A(z)$ near 1 is of the form

$$A(z) = C(1-z)^\alpha + O\left((1-z)^\beta\right) \qquad \text{with} \qquad \alpha \in \mathbb{R}\setminus\mathbb{N}, \quad \alpha < \beta \quad \text{and} \quad C \neq 0,$$

Transfer Theorem [11] gives that $a_n \sim \frac{C}{\Gamma(-\alpha)} n^{-\alpha-1}$, where Γ is Euler's Gamma function, the analytic continuation of $s \mapsto \int_0^\infty t^{s-1}e^{-t}dt$.

For more information on analytic combinatorics techniques, the reader is referred to the comprehensive book by Flajolet and Sedgewick [11].

4.2 Recognizing the empty word

This section is devoted to the proof of Theorem 3 which gives the probability that a regular expression of a given size recognizes the empty word.

Let r_n denote the probability that a size n regular expression does not recognize ε, with the convention $r_0 = 0$:

$$r_n = \sum_{\substack{T\in\mathcal{T}_n \\ \varepsilon\notin L(T)}} p(T).$$

▶ **Theorem 3.** *A large random regular expression recognizes the empty word with high probability. More precisely, in the BST-like model, the probability that a size n regular expression does not recognize ε is asymptotically equivalent to*

$$r_n \sim \frac{C}{n^q} \qquad \textit{with} \qquad C = \frac{(1-p_\varepsilon)}{e^{1-q}\Gamma(1-q)}\left(1 - \int_0^1 \frac{e^{(1-q)t}(1-t)^{1-q}-1}{t^2}dt\right).$$

Using basic computations, one can establish the following lemma from Equation (2):

▶ **Lemma 4.** *The sequence $(r_n)_{n\in\mathbb{N}}$ satisfies $r_1 = 1-p_\varepsilon$, $r_2 = 0$ and for any $n \geq 1$,*

$$r_{n+2} = \frac{1-q}{n}\sum_{\ell=1}^{n} r_\ell.$$

Let $R(z) = \sum_{n\in\mathbb{N}} r_n z^n$, with $r_0 = 0$, denote the generating function associated to the sequence $(r_n)_{n\in\mathbb{N}}$. For all $n \in \mathbb{N}$, since it is as a probability, r_n is in $[0,1]$; then $R(z)$ is analytic at 0 and its radius of convergence is at least 1.

▶ **Lemma 5.** *The generating function $R(z)$ satisfies the following differential equation*

$$z\frac{d}{dz}R(z) - \frac{(1-q)z^2 - 2z + 2}{1-z}R(z) + (1-p_\varepsilon)z = 0.$$

Proof. (Sketch) Multiply the general formula of Lemma 4 by nz^{n+2} and sum for $n \geq 1$. Then identify the expressions of the power series $R(z)$ and $\frac{d}{dz}R(z)$. ◀

▶ **Proposition 6.** Let g be the function defined by

$$g(z) = \frac{e^{(1-q)z}(1-z)^{1-q} - 1}{z^2}.$$

The function $g(z)$ has a false pole at zero, that can be removed, and one has

$$R(z) = (1-p_\varepsilon)\left(1 - z\int_0^z g(t)dt\right) ze^{(q-1)z}(1-z)^{q-1}.$$

Proof. The formula is obtained by the variation-of-constants method. Once stated, one can also directly verify that it satisfies the differential equation of Lemma 5 with the same initial conditions as $R(z)$. ◀

of Theorem 3. The proof is an application of analytic combinatorics techniques, and more precisely of singularity analysis of generating functions (see [11, Ch. VI]).

The function $g(z) = z^{-2}(e^{(1-q)z}(1-z)^{1-q} - 1)$ has its unique dominant singularity at 1 where we have:

$$g(z) = -1 + e^{1-q}(1-z)^{1-q} + O((1-z)).$$

Hence, by Singular Integration Theorem [11, p. 420], the antiderivative of g satisfies near 1 the following development:

$$\begin{aligned}\int_0^z g(t)dt &= -1 + O(1-z) - \frac{e^{1-q}}{2-q}(1-z)^{2-q} + \int_0^1 (g(t)+1)dt + O((1-z)^2)\\ &= \int_0^1 g(t)dt + O(1-z).\end{aligned}$$

Hence, $R(z)$ has its unique dominant singularity at 1 too, and near 1 one has

$$R(z) = (1-p_\varepsilon)e^{q-1}\left(1 - \int_0^1 g(t)\right)(1-z)^{q-1} + O((1-z)^q).$$

Using Transfer Theorem [11, p. 393], we obtain

$$r_n \sim \frac{(1-p_\varepsilon)e^{q-1}\left(1-\int_0^1 g(t)\right)}{\Gamma(1-q)} n^{-q},$$

concluding the proof.

◀

4.3 The average size of `First` is linear

In this section, we establish the following theorem. Some of the proofs are omitted, since they are similar to those of Theorem 3.

▶ **Theorem 7.** *The average size of* `First` *for a size n regular expression, according to the BST-like model, is asymptotically equivalent to $K\,n$, for some real constant $K \in]0,1[$.*

Let f_n be the average cardinality of `First` for regular expressions of size n:

$$f_n = \sum_{T\in\mathcal{T}_n} |\,\texttt{First}\,(T)\,| \cdot p(T).$$

Note that, by symmetry, f_n is also the average size of `Last` for regular expressions in $\mathcal{T}_n$.

▶ **Lemma 8.** *The sequence* $(f_n)_{n\in\mathbb{N}}$ *satisfies* $f_1 = f_2 = 1 - p_\varepsilon$ *and for any* $n \geq 1$,

$$f_{n+2} = qf_{n+1} + \frac{2(1-q)}{n}\sum_{\ell=1}^{n} f_\ell - \frac{1-q}{2n}\sum_{\ell=1}^{n} r_\ell f_{n+1-\ell}.$$

Let $F(z) = \sum_{n\in\mathbb{N}} f_n z^n$, with $f_0 = 0$, be the generating function associated to the sequence $(f_n)_{n\in\mathbb{N}}$.

▶ **Lemma 9.** *The generating function* $F(z)$ *satisfies the following differential equation*

$$z(1-qz)\frac{d}{dz}F(z) - \left(2 - qz + \frac{2(1-q)z^2}{1-z} - \frac{1-q}{2}zR(z)\right)F(z) + (1-p_\varepsilon)z = 0.$$

Solving this equation, we obtain the following proposition.

▶ **Proposition 10.** Let h be the function defined by

$$h(z) = \frac{(1-z)^2}{z^2(1-qz)^{2/q}} - \frac{1}{z^2}.$$

The function $h(z)$ has a false pole at zero, which can be removed, and one has

$$F(z) = \frac{z(1-qz)^{2/q-1}}{(1-z)^2}(1-p_\varepsilon)\exp\left(-\frac{1-q}{2}\int_0^z \frac{R(t)}{1-qt}dt\right)\left(1 + (1-q)z - z\int_0^z h(t)dt\right).$$

The proof of Theorem 7 is an analysis of $F(z)$ near its unique dominant singularity 1, using our result on $R(z)$. We obtain that

$$F(z) = \frac{K}{(1-z)^2}\left(1 + O\left((1-z)^q\right)\right) \quad \text{and} \quad f_n \sim K\, n,$$

with

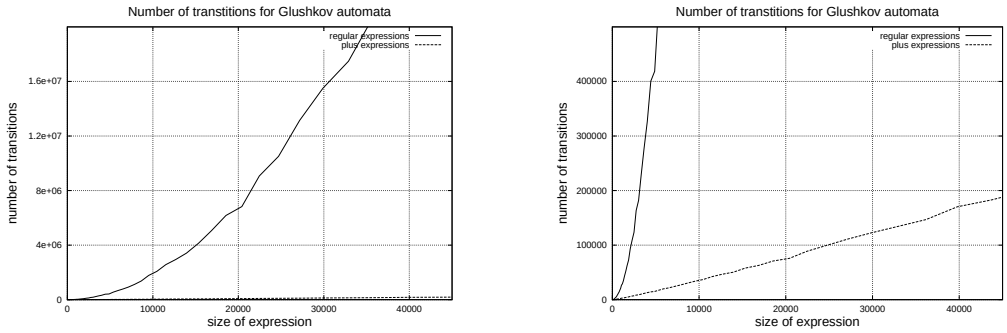
$$K = (1-p_\varepsilon)(1-q)^{2/q-1}\exp\left(-\frac{1-q}{2}\int_0^1 \frac{R(t)}{1-qt}dt\right)\left(2 - q - \int_0^1 h(t)dt\right).$$

5 Conclusion and perspectives

In this article, we analyzed the average size of Glushkov automata associated to random regular expressions, in the BST-like model. Using analytic combinatorics techniques, we proved that, unlike in the uniform case, the average number of transitions in an automaton is quadratic with respect to the size of the expression.

We implemented the procedure `RE(n)` given in Section 2.2 in order to confirm empirically our theoretical results. One of these experiments is displayed by Figure 5 (plain line). The x-axis represents the size of the regular expressions and the y-axis represents the number of transitions in the corresponding Glushkov automata. The dotted line corresponds to an other bench of experiments, involving a different kind of regular expressions, where the Kleene Star operator $*$ (reflexive and transitive closure of the concatenation) has been replaced by a $+$ operator (only transitive closure of the concatenation). Considering the classical regular expressions, the quadratic behavior clearly appears on Figure 5, whereas it seems to be linear for expressions using only the $+$ operator.

One can reasonably expect to prove the linear behavior observed in Figure 5b, using the techniques of the present paper combined with those of [17]. This seems to be confirmed by the calculations we have already performed. A natural extension of this work is therefore to complete this proof. In a different direction, the average analysis of other constructions related to Glushkov automata, could be considered. Among them are the Follow automaton by Ilie and Yu [13] and Antimirov automaton [2], which are both quotients of Glushkov automaton (see [4]).



(a) **(b)**

Figure 5 Number of transitions of Glushkov automata with respect to the size of expressions defined on the alphabet $A = \{a, b\}$, with parameters $q = \frac{1}{3}$, $p_\varepsilon = \frac{1}{100}$ and $p(a) = p(b)$. Note that (a) and (b) display the same data, but with different scales.

References

1 Alfred V. Aho, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: principles, techniques, and tools*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1986.

2 Valentin Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.

3 Gérard Berry and Ravi Sethi. From regular expressions to deterministic automata. *Theoretical Computer Science*, 48(1):117–126, 1986.

4 Jean-Marc Champarnaud and Djelloul Ziadi. Canonical derivatives, partial derivatives and finite automaton constructions. *Theoretical Computer Science*, 289(1):137 – 163, 2002.

5 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press, Cambridge, MA, second edition, 2001.

6 Marco Daniele, Fausto Giunchiglia, and Moshe Y. Vardi. Improved automata generation for Linear Temporal Logic. In Nicolas Halbwachs and Doron Peled, editors, *CAV*, volume 1633 of *Lecture Notes in Computer Science*, pages 249–260. Springer, 1999.

7 Luc Devroye. A note on the height of binary search trees. *Journal of the ACM*, 33(3):489–498, 1986.

8 Michael Drmota. An analytic approach to the height of binary search trees. *Journal of the ACM*, 50:89–119, 2001.

9 Philippe Duchon, Philippe Flajolet, Guy Louchard, and Gilles Schaeffer. Boltzmann samplers for the random generation of combinatorial structures. *Combinatorics, Probability and Computing*, 13(4–5):577–625, 2004.

10 Philippe Flajolet and Andrew Odlyzko. The average height of binary trees and other simple trees. *The Journal of Computer and System Sciences*, 25(2):171–213, 1982.

11 Philippe Flajolet and Robert Sedgewick. *Analytic Combinatorics*. Cambridge University Press, 2009.

12 Victor Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16:1–53, 1961.

13 Lucian Ilie and Sheng Yu. Follow automata. *Information and Computation*, 186(1):140–162, 2003.

14 Stephen Cole Kleene. Representation of events in nerve nets and finite automata. *Automata Studies, Annals of Mathematics Studies*, 36, 1956.

15 Conrado Martínez. *Statistics Under the BST Model*. PhD thesis, UPC, Spain, 1992.

16 Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *IRE Transactions on Electronic Computers*, 9:39–47, 1960.

17 Cyril Nicaud. On the average size of Glushkov's automata. In Adrian Horia Dediu, Armand-Mihai Ionescu, and Carlos Martín-Vide, editors, *LATA*, volume 5457 of *Lecture Notes in Computer Science*, pages 626–637. Springer, 2009.

18 John M. Robson. The height of binary search trees. *Australian Computer Journal*, 11(4):151–153, 1979.

19 Heikki Tauriainen and Keijo Heljanko. Testing SPIN's LTL formula conversion into Büchi automata with randomly generated input. In Klaus Havelund, John Penix, and Willem Visser, editors, *SPIN*, volume 1885 of *Lecture Notes in Computer Science*, pages 54–72. Springer, 2000.

Beyond Hyper-Minimisation—Minimising DBAs and DPAs is NP-Complete*

Sven Schewe

University of Liverpool

Abstract

In this paper we study the problem of minimising deterministic automata over finite and infinite words. Deterministic finite automata are the simplest devices to recognise regular languages, and deterministic Büchi, Co-Büchi, and parity automata play a similar role in the recognition of ω-regular languages. While it is well known that the minimisation of deterministic finite and weak automata is cheap, the complexity of minimising deterministic Büchi and parity automata has remained an open challenge. We establish the NP-completeness of these problems. A second contribution of this paper is the introduction of almost equivalence, an equivalence class for strictly between language equivalence for deterministic Büchi or Co-Büchi automata and language equivalence for deterministic finite automata. Two finite automata are almost equivalent if they, when used as a monitor, provide a different answer only a bounded number of times in any run, and we call the minimal such automaton relatively minimal. Minimisation of DFAs, hyper-minimisation, relative minimisation, and the minimisation of deterministic Büchi (or Co-Büchi) automata are operations of increasing reduction power, as the respective equivalence relations on automata become coarser from left to right. Besides being a natural equivalence relation for finite automata, almost equivalence is language preserving for weak automata, and can therefore also be viewed as a generalisation of language equivalence for weak automata to a more general class of automata. From the perspective of Büchi and Co-Büchi automata, we gain a cheap algorithm for state-space reduction that also turns out to be beneficial for further heuristic or exhaustive state-space reductions put on top of it.

1998 ACM Subject Classification F.1.1 Models of Computation

Keywords and phrases Automata Theory, Complexity, Büchi Automata, Parity Automata

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.400

1 Introduction

The minimisation of deterministic finite automata (DFAs) is a classic problem with an efficient solution [8, 9]. This paper was originally written with only the question in mind of whether or not a similar result can be obtained for deterministic automata over infinite words. Is their minimisation tractable? For weak automata, the answer is known to be positive [12], which seems to encourage a quest for a tractable solution for Büchi, Co-Büchi, and parity automata as well. However, it turns out that their minimisation is intractable (NP-complete).

This raised the question whether there are natural tractable problems between the minimisation of DFAs and deterministic Büchi automata (DBAs) or deterministic Co-Büchi automata (DCAs). The hyper-minimisation of deterministic automata [2, 1, 7] is such an

* This work was partly supported by the Engineering and Physical Science Research Council (EPSRC) through the grant EP/H046623/1 'Synthesis and Verification in Markov Game Structures'. An extended version is available as a technical report [21], which also contains the omitted proofs.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 400–411

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

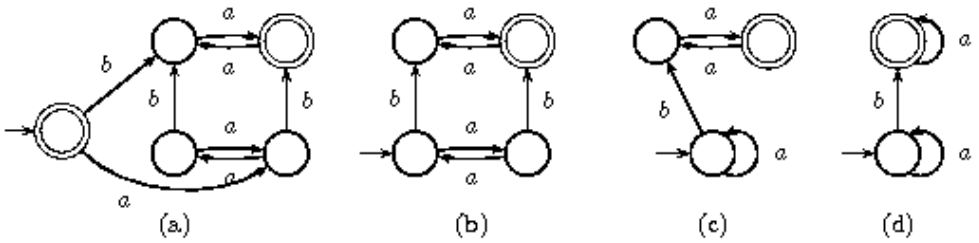


Figure 1 Figure 1a shows a minimal DFA $\mathcal{A}$ over the two letter alphabet $\{a, b\}$. Figure 1b shows a hyperminisation of $\mathcal{A}$, Figure 1c shows a minimal almost equivalent automaton to $\mathcal{A}$, and Figure 1d shows a minimal language equivalent DBA to $\mathcal{A}$. (Neither hyperminimal nor relative minimal automata need to be unique.) Hyperminimal automata are the minimal automata with a finite symmetric language difference to the source automaton [2, 1, 7]; they may only differ from the minimal automaton in the preamble. (The non-trivial SCCs of the automaton reachable from the initial state.) Minimal almost equivalent automata only guarantee that the symmetrical difference intersected with the prefixes of every infinite word are finite. (In both cases, finite implies bounded by the number of states of the automaton.) For weak automata—automata whose language is equivalent when read as DBA or DCA—a minimal almost equivalent automaton is also a minimal weak automaton.

example: If we minimise a DFA while allowing for a finite symmetrical difference between the language of the source and target automaton, we may be rewarded by a smaller automaton.

We introduce a second relaxation, almost equivalence, where we require that acceptance differs only on finitely many prefixes of every infinite word. This provides the guarantee that, on each infinite run, the result is equivalent in almost all positions (cf. Figure 1), which is not only interesting in itself, but can also be viewed as a generalisation of the minimisation problem of weak automata [12] to a more general class.

This is a natural notion of almost equivalence on DFAs, which also forms a promising basis for state-space reduction of Büchi and Co-Büchi automata. Different to the NP-completeness of minimising Büchi and Co-Büchi automata, we show that finding a minimal almost equivalent DFA is cheap. It is also a useful starting point for a state-space reduction of a DBA or DCA $\mathcal{A}$, because minimisation with respect to almost equivalence (like minimisation and hyper-minimisation) of $\mathcal{A}$ when read as a DFA are language preserving.

The algorithm we develop for finding a minimal almost equivalent DFA can be strengthened by using language equivalence on $\mathcal{A}$ (when read as a Büchi or Co-Büchi automata) in the algorithm, which provides for a smaller—yet still language equivalent—target automaton. This automaton has the interesting property that one can focus on its strongly connected components (SCCs) in isolation when reducing its state-space further.

While the NP-completeness of the minimisation problem of DBAs, DCAs, and deterministic parity automata (DPAs) seems to rule out the use of state-space reduction on large scale problems, this reduction technique therefore suggests that one might often get far on the way of reducing the state-space without having to pay a high price, while getting for free a division of the remaining potential parts of the automaton for further reduction.

This is fortunate, because the standard verification technique for the verification of Markov decision processes against LTL specifications [3] as well as the synthesis of distributed systems from LTL specifications [19, 16, 17, 22, 15, 20] require working with these deterministic ω-automata, and techniques for the minimisation, or, indeed, for the state-space reduction of the automata involved are more than welcome. The argument in favour of such reductions becomes even stronger for algorithms that synthesise distributed systems [18, 11, 13, 23, 6],

where deterministic automata occur in various steps of the construction.

2 Deterministic Automata

2.1 ω-Automata

Parity automata are word automata that recognise ω-regular languages over finite set of symbols. A *deterministic parity automaton* is a tuple $\mathcal{P} = (\Sigma, Q, q_0, \delta, \pi)$, where

- Σ denotes a finite set of symbols,
- Q denotes a finite set of states,
- $q_0 \in Q_+$ with $Q_+ = Q \dot\cup \{\bot, \top\}$ denotes a designated initial state,
- $\delta : Q_+ \times \Sigma \to Q_+$ is a function that maps pairs of states and input letters to either a new state, or to $\bot$ (false, immediate rejection, blocking) or $\top$ (true, immediate acceptance)[1], such that $\delta(\top, \sigma) = \top$ and $\delta(\bot, \sigma) = \bot$ hold for all $\sigma \in \Sigma$, and
- $\pi : Q_+ \to P \subset \mathbb{N}$ is a priority function that maps states to natural numbers (mapping $\bot$ and $\top$ to an odd and even number, respectively), called their *priority*. (They are often referred to as colours.)

Parity automata read infinite input words $\alpha = a_0 a_1 a_2 \ldots \in \Sigma^\omega$. (As usual, $\omega = \mathbb{N}_0$ denotes the non-negative integers.) Their acceptance mechanism is defined in terms of runs: The unique run $\rho = r_0 r_1 r_2 \ldots \in {Q_+}^\omega$ of $\mathcal{P}$ on α is the ω-word that satisfies $r_0 = q_0$ and, for all $i \in \omega$, $r_{i+1} = \delta(r_i, a_i)$. A run is called *accepting* if the highest number occurring infinitely often in the infinite sequence $\pi(r_0)\pi(r_1)\pi(r_2)\ldots$ is even, and *rejecting* if it is odd. An ω-word is *accepted* by $\mathcal{P}$ if its run is accepting. The set of ω-words accepted by $\mathcal{P}$ is called its *language*, denoted $\mathcal{L}(\mathcal{P})$.

We assume without loss of generality that $\max P \leq |Q| + 1$. (If a priority $p \succeq 2$ does not exist, we can reduce the priority of all states whose priority is strictly greater than p by 2 without affecting acceptance.)

Deterministic Büchi and Co-Büchi automata—abbreviated DBAs and DCAs—are DPAs where the image of the priority function π is contained in $\{1, 2\}$ and $\{2, 3\}$, respectively. In both cases, the automaton is often denoted $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$, where $F \subseteq Q_+$ denotes those states with priority 2. The states in F are also called *final* or *accepting states*, while the remaining states $Q_+ \setminus F$ are called *rejecting* states.

2.2 Finite Automata

Finite automata are word automata that recognise the regular languages over finite set of symbols. A *deterministic finite automaton* (DFA) is a tuple $\mathcal{F} = (\Sigma, Q, q_0, \delta, F)$, where Σ, Q, q_0, and δ are defined a for DPAs, and $F \subseteq Q \dot\cup \{\top\}$ is a set of *final states* that contains $\top$ (but not $\bot$).

Finite automata read finite input words $\alpha = a_0 a_1 a_2 \ldots a_n \in \Sigma^*$. Their acceptance mechanism is again defined in terms of runs: The unique run $\rho = r_0 r_1 r_2 \ldots r_{n+1} \in {Q_+}^+$ of $\mathcal{F}$ on α is the word that satisfies $r_0 = q_0$ and, for all $i \leq n$, $r_{i+1} = \delta(r_i, a_i)$. A run is called *accepting* if it ends in a final state (and *rejecting* otherwise), a word is *accepted* by $\mathcal{F}$ if its run is accepting, and the set of words accepted by $\mathcal{F}$ is called its *language*, denoted $\mathcal{L}(\mathcal{F})$.

[1] The question whether or not an automaton can immediately accept or reject is a matter of taste. Often, immediate rejection is covered by allowing δ to be partial while there is no immediate acceptance. For technical convenience, we allow both, but treat $\top$ and $\bot$ as accepting and rejecting sink states, respectively.

2.3 Automata Transformations & Conventions

For a deterministic automaton $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ or $\mathcal{A} = (\Sigma, Q, q_0, \delta, \pi)$ and a state $q \in Q_+$, we denote with $\mathcal{A}_q = (\Sigma, Q, q, \delta, F)$ or $\mathcal{A}_q = (\Sigma, Q, q, \delta, \pi)$, respectively, the automaton resulting from $\mathcal{A}$ by changing the initial state to q. We also read finite automata at times as Büchi (or Co-Büchi) automata and Büchi (or Co-Büchi) automata as finite automata in the constructions, and let DFAs run on infinite words where this is convenient and its meaning is clear in the context.

Automata define a directed graph whose unravelling from the initial state defines the possible runs. For an automaton $\mathcal{A} = (\Sigma, Q, q_0, \delta, F)$ or $\mathcal{A} = (\Sigma, Q, q_0, \delta, \pi)$, this is the directed graph (Q_+, T) with $T = \{(p, q) \in Q_+ \times Q_+ \mid \exists \sigma \in \Sigma.\ \delta(p, \sigma) = q\}$. When referring to the reachable states (which always means reachable from the initial state) and SCCs of an automaton, this refers to this graph.

2.4 Emptiness and Equivalence

A DPA is called *empty* if its language is empty and *universal* if it accepts every word $\alpha \in \Sigma^\omega$. For two automata $\mathcal{P}^1 = (\Sigma, Q_1, q_0^1, \delta_1, \pi_1)$ and $\mathcal{P}^2 = (\Sigma, Q_2, q_0^2, \delta_2, \pi_2)$, two states $q_1 \in Q_1$ and $q_2 \in Q_2$ are called *equivalent* if $\mathcal{L}(\mathcal{P}^1_{q_1}) = \mathcal{L}(\mathcal{P}^2_{q_2})$. (Equivalence of states naturally extends to the same automaton, as $\mathcal{P}^1$ and $\mathcal{P}^2$ are not necessarily different.) Two automata are equivalent if their initial states are equivalent. (Or, likewise, if they recognise the same language.)

Emptiness, universality, and equivalence of parity, Büchi, and Co-Büchi automata is computationally easy:

▶ **Theorem 1.** *Language inclusion, equivalence, emptiness, and universality of parity, Büchi, and Co-Büchi automata and their co-problems are NL-complete.*

3 Minimising Büchi and Parity Automata is NP-Complete

In this section we show that the minimisation of deterministic Büchi, Co-Büchi, and parity automata are NP-complete problems. This is in contrast to the tractable minimisation of finite [8] and weak automata [12].

The hardest part of the NP-completeness proof is a reduction from the problem of finding a minimal vertex cover of a graph to the minimisation of deterministic Büchi automata. For this reduction, we first define the characteristic language of a simple connected graph. For technical convenience we assume that this graph has a distinguished initial vertex.

We show that the states of a deterministic Büchi automaton that recognises this characteristic language must satisfy side-constraints, which imply that it has at least $2n + k$ states, where n is the number of vertices of the graph, and k is the size of its minimal vertex cover. We then show that, given a vertex cover of size k, it is simple to construct a deterministic Büchi automaton of size $2n + k$ that recognises the characteristic language of this graph. (It can be constructed in linear time and logarithmic space.) Furthermore, we show that minimising the automaton defined by the trivial vertex cover can be used to determine a minimal vertex cover for this graph, which concludes the reduction.

We call a non-trivial ($|V| > 1$) simple connected graph $\mathcal{G}_{v_0} = (V, E)$ with a distinguished initial vertex $v_0 \in V$ *nice*. As a warm-up, we have to show that the restriction to nice graphs leaves the problem of finding a minimal vertex cover NP-complete.

▶ **Lemma 2.** *The problem of checking whether a nice graph $\mathcal{G}_{v_0}$ has a vertex cover of size k is NP-complete.*

Proof. As a special case of the vertex cover problem, it is in NP, and the problem of finding a vertex cover of size k for a graph (V, E) can be reduced to the problem of checking if the nice graph $\mathcal{G}_v = (V \dot\cup \{v, v'\}, E \cup \big\{\{w, v\} \mid w \in V \dot\cup \{v'\}\big\}$ has a vertex cover of size $k+1$: A vertex cover of $\mathcal{G}_v$ must contain a vertex cover of (V, E) and v or v', and a vertex cover of (V, E) plus v is a vertex cover of $\mathcal{G}_v$. ◀

We define the *characteristic language* $\mathcal{L}(\mathcal{G}_{v_0})$ of a nice graph $\mathcal{G}_{v_0}$ as the ω-language over $V_\natural = V \dot\cup \{\natural\}$ (where $\natural$ indicates a stop of the evaluation in the next step—it can be read 'stop') consisting of

1. all ω-words of the form $v_0{}^* v_1{}^+ v_2{}^+ v_3{}^+ v_4{}^+ \ldots \in V^\omega$ with $\{v_{i-1}, v_i\} \in E$ for all $i \in \mathbb{N}$, (words where $v_0, v_1, v_2, \ldots$ form an infinite path in $\mathcal{G}_{v_0}$), and
2. all ω-words starting with $v_0{}^* v_1{}^+ v_2{}^+ \ldots v_n{}^+ \natural v_n \in V_\natural{}^*$ with $n \in \mathbb{N}_0$ and $\{v_{i-1}, v_i\} \in E$ for all $i \in \mathbb{N}$. (Words where $v_0, v_1, v_2, \ldots, v_n$ form a finite—and potentially trivial—path in $\mathcal{G}_{v_0}$, followed by a $\natural$ sign, followed by the last vertex of the path $v_0, v_1, v_2, \ldots, v_n$.)

We call the ω-words in (1) *trace-words*, and those in (2) *$\natural$-words*. The trace-words are in V^ω, while the $\natural$-words are in ${V_\natural}^\omega \setminus V^\omega$.

Let $\mathcal{B}$ be a deterministic Büchi automaton that recognises the characteristic language of $\mathcal{G}_{v_0} = (V, E)$. We call a state of $\mathcal{B}$

- a *v-state* if it can be reached upon an input word $v_0{}^* v_1{}^+ v_2{}^+ \ldots v_n{}^+ \in V_\natural{}^*$, with $n \in \mathbb{N}_0$ and $\{v_{i-1}, v_i\} \in E$ for all $i \in \mathbb{N}$, that ends in $v = v_n$ (in particular, the initial state of $\mathcal{B}$ is a v_0-state), and
- a *$v\natural$-state* if it can be reached from a v-state upon reading a $\natural$ sign.

We call the union over all v-states the set of *vertex-states*, and the union over all $v\natural$-states the set of $\natural$-states.

▶ **Lemma 3.** *Let $\mathcal{G}_{v_0} = (V, E)$ be a nice graph with initial vertex v_0, and let $\mathcal{B} = (V, Q, q_0, \delta, F)$ be a deterministic Büchi automaton that recognises the characteristic language of $\mathcal{G}_{v_0}$. Then (1) the vertex- and $\natural$-states of $\mathcal{B}$ are disjoint, and, for all $v, w \in V$ with $v \neq w$, (2) the v-states and w-states and (3) the $v\natural$- and $w\natural$-states are disjoint. For each vertex $v \in V$, there is (4) a $v\natural$-state and (5) a rejecting v-state, and (6), for every edge $\{v, w\} \in E$, there is an accepting v-state or an accepting w-state.*

Proof. 1. Let $q_v^\natural$ be a $v\natural$-state and q a vertex-state. As $\mathcal{B}$ recognises $\mathcal{L}(\mathcal{G}_{v_0})$, $\mathcal{B}_{q_v^\natural}$ must accept v^ω, while $\mathcal{B}_q$ must reject it.

2. Let q_v be a v-state and let q_w be a w-state with $v \neq w$. As $\mathcal{B}$ recognises $\mathcal{L}(\mathcal{G}_{v_0})$, $\mathcal{B}_{q_v}$ must accept $\natural v^\omega$, while $\mathcal{B}_{q_w}$ must reject it.

3. Let $q_v^\natural$ be a $v\natural$-state and let $q_w^\natural$ be a $w\natural$-state with $v \neq w$. As $\mathcal{B}$ recognises $\mathcal{L}(\mathcal{G}_{v_0})$, $\mathcal{B}_{q_v^\natural}$ must accept v^ω, while $\mathcal{B}_{q_w^\natural}$ must reject it.

4. As $\mathcal{G}_{v_0}$ is connected, there is, for every $v \in V$, a path $v_0 v_1 v_2 \ldots v$ in $\mathcal{G}_{v_0}$, and the state reached by $\mathcal{B}$ upon reading $v_1 v_2 \ldots v\natural$ is a $v\natural$-state.

5. As $\mathcal{G}_{v_0}$ is connected, there is, for every $v \in V$, a path $v_0 v_1 v_2 \ldots v$ in $\mathcal{G}_{v_0}$. After reading $v_1 v_2 \ldots v$, $\mathcal{B}$ is in a v-state. $\mathcal{B}$ remains in v-states if it henceforth reads v's. (Note that the automaton cannot block/reject immediately, as it should accept a continuation $\natural v^\omega$ at any time.) As the word is rejecting, almost all states in the run of the automaton are rejecting v-states.

6. Let us consider an arbitrary edge $\{v, w\}$. As $\mathcal{G}_{v_0}$ is connected, there is a path from $v_0 v_1 v_2 \ldots v$ in $\mathcal{G}_{v_0}$, and $v_1 v_2 \ldots v(wv)^\omega$ is in $\mathcal{L}(\mathcal{G}_{v_0})$; the run of $\mathcal{B}$ on this ω-word is therefore accepting. As almost all states in this accepting run are v-states or w-states, there must be an accepting v-state or an accepting w-state.

◀

The sixth claim implies that the set C of vertices with an accepting vertex-state is a vertex cover of $\mathcal{G}_{v_0} = (V, E)$. It is also clear that $\mathcal{B}$ has at least $|V|$ rejecting vertex-states, $|C|$ accepting vertex-states, and $|V|$ $\natural$-states:

▶ **Corollary 4.** *For a deterministic Büchi automaton that recognises the characteristic language of a nice graph $\mathcal{G}_{v_0} = (V, E)$ with initial vertex v_0, the set $C = \{v \in V \mid$ there is an accepting v-state$\}$ is a vertex cover of $\mathcal{G}_{v_0}$, and $\mathcal{B}$ has at least $2|V| + |C|$ states.* ◀

It is not hard to define, for a given nice graph $\mathcal{G}_{v_0} = (V, E)$ with vertex cover C, a Büchi automaton $\mathcal{B}_C^{\mathcal{G}_{v_0}} = (V_\natural, (V \times \{r, \natural\}) \dot\cup (C \times \{a\}), (v_0, r), \delta, (C \times \{a\}) \dot\cup \{\top\})$ with $2|V| + |C|$ states that recognises the characteristic language of $\mathcal{G}_{v_0}$: We simply choose

- $\delta\big((v, r), v'\big) = (v', a)$ if $\{v, v'\} \in E$ and $v' \in C$,
  $\delta\big((v, r), v'\big) = (v', r)$ if $\{v, v'\} \in E$ and $v' \notin C$,
  $\delta\big((v, r), v'\big) = (v, r)$ if $v = v'$,
  $\delta\big((v, r), v'\big) = (v, \natural)$ if $v' = \natural$, and
  $\delta\big((v, r), v'\big) = \bot$ otherwise;
- $\delta\big((v, a), v'\big) = \delta\big((v, r), v'\big)$, and
- $\delta\big((v, \natural), v\big) = \top$ and $\delta\big((v, \natural), v'\big) = \bot$ for $v' \neq v$.

$\mathcal{B}_C^{\mathcal{G}_{v_0}}$ simply has one $v\natural$-state for each vertex $v \in V$ of $\mathcal{G}_{v_0}$, one accepting v-state for each vertex in the vertex cover C, and one rejecting v-vertex for each vertex $v \in V$ of $\mathcal{G}_{v_0}$. It moves to the accepting copy of a vertex state v only upon taking an edge to v, but not on a repetition of v.

▶ **Lemma 5.** *For a nice graph $\mathcal{G}_{v_0} = (V, E)$ with initial vertex v_0 and vertex cover C, $\mathcal{B}_C^{\mathcal{G}_{v_0}}$ recognises the characteristic language of $\mathcal{G}_{v_0}$.*

Proof. To show $\mathcal{L}(\mathcal{B}_C^{\mathcal{G}_{v_0}}) \subseteq \mathcal{L}(\mathcal{G}_{v_0})$, let us consider an ω-word α accepted by $\mathcal{B}_C^{\mathcal{G}_{v_0}}$. Then it is either eventually accepted immediately when reading a v from a state $(v, \natural)$, or by seeing accepting states in $C \times \{a\}$ infinitely many times. By the construction of $\mathcal{B}_C^{\mathcal{G}_{v_0}}$, α must be a $v\natural$-word in the first case, and a trace-word in the latter.

To show $\mathcal{L}(\mathcal{B}_C^{\mathcal{G}_{v_0}}) \supseteq \mathcal{L}(\mathcal{G}_{v_0})$, it is apparent that $\natural$-words are accepted immediately after reading the initial sequence that makes them $\natural$-words, while a trace-word ${v_0}^{i_0-1} {v_1}^{i_1} {v_2}^{i_2} {v_3}^{i_3} \ldots \in V^\omega$ with $i_j \in \mathbb{N}$ and $\{v_j, v_{j+1}\} \in E$ for all $j \in \omega$, has the run $\rho = (v_0, r)^{i_0}(v_1, p_1)(v_1, r)^{i_1-1}(v_2, p_2)(v_2, r)^{i_2-1}(v_3, p_3)\ldots$, with $p_i = a$ (and hence (v_i, p_i) accepting) if v_i in C. As C is a vertex cover, this is at least the case for every second index. (There is no $n \in \mathbb{N}$ with $\{v_n, v_{n+1}\} \cap C = \emptyset$.) ρ therefore contains infinitely many accepting states. ◀

Corollary 4 and Lemma 5 immediately imply:

▶ **Corollary 6.** *Let C be a minimal vertex cover of a nice graph $\mathcal{G}_{v_0} = (V, E)$. Then $\mathcal{B}_C^{\mathcal{G}_{v_0}}$ is a minimal deterministic Büchi automaton that recognises the characteristic language of $\mathcal{G}_{v_0}$.* ◀

From here, it is a small step to the main theorem of this section:

▶ **Theorem 7.** *The problem of whether there is, for a given deterministic Büchi automaton, a language equivalent Büchi automaton with at most n states is NP-complete.*

Proof. For containment in NP, we can simply use non-determinism to guess such an automaton. Checking that it is language equivalent is then in NL by Theorem 1.

By Corollary 6, we can reduce checking if a nice graph G_v with m vertices has a vertex cover of size k to checking if the deterministic Büchi automaton $\mathcal{B}_V^{G_v}$—which has $3m$ states and is easy to construct (in deterministic logspace)—has a language equivalent Büchi automaton with $2m + k$ states. As the problem we reduced from is NP-complete by Lemma 2, this concludes the reduction. ◀

As minimising Co-Büchi automata coincides with minimising the dual Büchi automata, the similar claim holds for Co-Büchi automata.

▶ **Corollary 8.** *The problem of whether there is, for a given deterministic Co-Büchi automaton, a language equivalent Co-Büchi automaton with at most n states is NP-complete.* ◀

The problem of minimising deterministic parity automata cannot be easier than the problem of minimising Büchi automata, and the 'in NP' argument that we can simply guess a language equivalent DPA and then inexpensively check correctness (by Theorem 1) extends to parity automata.

▶ **Corollary 9.** *The problem if there is, for a given parity automaton, a language equivalent parity automaton with n states is NP-complete.* ◀

Note that, while there is a minimal number of priorities required for every language, the number of states cannot be reduced by increasing the number of priorities, and minimising the number of priorities can be done in polynomial time, changing only the priority functions [14, 4].

4 Relative DFA Minimisation

Minimisation techniques for deterministic finite automata can be used to minimise deterministic Büchi and Co-Büchi automata. They are cheap—Hopcroft's algorithm works in time $\mathcal{O}(n \log n)$ [8]—and have proven to be powerful devices for state-space reduction. From a practical point of view, this invites—in the light of the intractability result for minimising deterministic Büchi and Co-Büchi automata—the question if such tractable minimisation techniques can be used for a space reduction of Büchi and Co-Büchi automata. From a theoretical point of view, this invites the question of whether there are interesting tractable minimisation problems between the minimisation (or hyper-minimisation [2, 1, 7]) of finite automata, and the minimisation of Büchi and Co-Büchi automata.

Both the theoretical and the practical question turn out to have a positive answer: An answer to the theoretical question is that we can define *almost equivalence* on automata and their states as a relation, where two automata or states are almost equivalent if their language intersected with the initial sequences of every omega word have finite difference. We show that a minimal almost equivalent automaton is easy to construct. Besides being interesting on their own account (for example, if we want to construct a monitor that errs only a bounded number of times for every input word), they are language preserving for deterministic Büchi and Co-Büchi automata. What is more, a minimal almost equivalent automata to a weak automaton (an automaton that recognises the same language as DBA and DCA) is a minimal language equivalent weak automaton.

From a practical point of view, the algorithm suggests an approximation that is valid for both Büchi and Co-Büchi automata. There is, however, a simple and apparent improvement of the algorithm when used for the minimisation of Büchi and Co-Büchi automata: Instead of almost equivalence of states, we can use language equivalence for Büchi or Co-Büchi automata, respectively. But the algorithm provides for more: It isolates the minimisation

problem within in the SCC. That is, both precise and approximative minimisation techniques can look into these simpler sub-structures.

While being language preserving when the DBA or DCA is read as a DFA is a sufficient criterion for language preservation of the automaton itself, it is by no means necessary. In this context it becomes apparent that the NP-completeness result of the previous section may not hint at the fact that state-space reduction for DBAs and DCAs is beyond price; one should rather take it as a hint that a high price might have to be paid for the *additional* benefit one can get from stronger state-space reductions than those for DFAs.

However, even if we consider DFAs, there is at time a desire for stronger reductions than language preserving minimisation. For this reason, hyper-minimisation, the problem of finding a minimal automaton with a finite symmetrical difference in its language, has been studied for DFAs [2, 1, 7]. In this section, we introduce relative minimisation where we seek a minimal automaton for which the symmetrical difference intersected with the initial sequences of every infinite word is bounded. The underlying notion of approximate equivalence is weaker than the f-equivalence used for hyper-minimisation, and in my opinion it is also more natural even for DFAs. (One is often not really interested in differences on words that one never observes.) It surely is the better starting point for minimising DBAs and DCAs. We develop a simple algorithm for relative minimisation, and discuss how it can be strengthened to approximate minimal DBAs or DCAs even better.

4.1 Almost Equivalence

For two (not necessarily different) DFAs $\mathcal{A}^1 = (\Sigma, Q_1, q_0^1, \delta_1, F_1)$ and $\mathcal{A}^2 = (\Sigma, Q_2, q_0^2, \delta_2, F_2)$, we call two states $q_1 \in Q_1$ and $q_2 \in Q_2$ *almost equivalent* if, for all ω-words $\alpha \in \Sigma^\omega$, it holds that for the runs $r_0^1 r_1^1 r_2^1 r_3^1 \ldots$ and $r_0^2 r_1^2 r_2^2 r_3^2 \ldots$ of $\mathcal{A}^1_{q_1}$ and $\mathcal{A}^2_{q_2}$ on α, membership of the states in the final states is equivalent almost everywhere ($\exists n \in \omega.\ \forall i \geq n.\ r_i^1 \in F_1 \Leftrightarrow r_i^2 \in F_2$). Two DFAs are called almost equivalent if their initial states are, and we extend these definitions to DBAs and DCAs.

Obviously, almost equivalence is a congruence and hence defines quotient classes on the states of automata. It is also easy to compute:

▶ **Lemma 10.** *Testing almost equivalence (or inequivalence) of two DFAs $\mathcal{A}$ and $\mathcal{B}$ is NL-complete, and the quotient class of a DFA $\mathcal{A}$ can be computed in time quadratic in the size of the automaton.*

Proof. It is simple to construct in deterministic logspace an automaton $\mathcal{A} \intercal \mathcal{B}$ whose states are ordered pairs of $\mathcal{A}$ and $\mathcal{B}$ states, with the pair of initial states of $\mathcal{A}$ and $\mathcal{B}$ as initial state, whose final states are the pairs of a final and a non-final state (where the final state might be an $\mathcal{A}$ or a $\mathcal{B}$ state). Two states q_a and q_b are obviously almost equivalent if, and only if, the language of $(\mathcal{A} \intercal \mathcal{B})_{(q_a,q_b)}$ is empty when read as a DBA, which is in NL by Theorem 1. For completeness, it is again easy to reduce the reachability problem of directed graphs to refuting almost equivalence of two automata.

This simple construction also caters for a quadratic deterministic algorithm for finding the quotients of almost equivalent states: We can construct $\mathcal{A} \intercal \mathcal{A}$ in quadratic time and find the SCCs in $\mathcal{A} \intercal \mathcal{A}$ in time linear in $\mathcal{A} \intercal \mathcal{A}$. Two states p, q are obviously either almost equivalent or one can reach a final state in a non-trivial SCC from (p, q) in $\mathcal{A} \intercal \mathcal{A}$, and these states can be computed in time linear in $\mathcal{A} \intercal \mathcal{A}$ by a simple fixed-point algorithm. ◀

4.2 Finding minimal almost equivalent automata is tractable

We call the problem of finding a minimal automaton almost equivalent to a DFA $\mathcal{A}$ *relative minimisation*. Besides the usefulness of relative minimisation for DFAs themselves, let us consider the usefulness of relative minimisation for the state-space reduction of deterministic Büchi and Co-Büchi automata.

▶ **Lemma 11.** *Two deterministic Büchi and Co-Büchi automata that are, when read as deterministic finite automata, almost language equivalent recognise the same language.*

We can therefore use the inexpensive DFA minimisation, hyper-minimisation (which in particular results in an almost equivalent automaton), and the newly introduced relative minimisation of DFAs for a state-space reduction of DBAs and DCAs. This provides the back-bone for efficient relative minimisation: To find, for a given DFA $\mathcal{A} = (\Sigma, Q', q_0', \delta'', F')$, a minimal deterministic automaton $\mathcal{D}$ that accepts an almost equivalent language, we execute the following algorithm:

▶ Construction 12. In a first step[2], we construct the minimal language equivalent automaton $\mathcal{B} = (\Sigma, Q, q_0, \delta, F)$ in quasi-linear time using Hopcroft's algorithm [8].

For $\mathcal{B}$, we then introduce a pre-order $(Q_+, \succeq)$ on the states of $\mathcal{B}$ such that (1) two states are equivalent if, and only if, they are in the same SCC of $\mathcal{B}$, (2) if p is reachable from q then $p \succeq q$, and (3) $\top$ and $\bot$ are bigger than all states in Q. (This can obviously be done in linear time.)

In a third step, we determine the quotient classes of almost equivalent states of $\mathcal{B}$, and pick, for each quotient class $[q]$, a representative $r_{[q]} \in [q]$ that is maximal with respect to $\succeq$ among the states almost equivalent to q.

We then construct an automaton $\mathcal{C} = (\Sigma, Q, r_{[q_0]}, \delta', F)$ by choosing the representative $r_{[q_0]}$ of the quotient $[q_0]$ of states almost equivalent to the initial state as new initial state, and changing all transitions that lead to states whose representative is bigger (with respect to $\succeq$) to the representatives of these states. That is, for $\delta(q, \sigma) = q'$, we get $\delta'(q, \sigma) = q'$ if $q \simeq r_{[q']}$ and $\delta'(q, \sigma) = r_{[q']}$ otherwise.

Finally, we minimise $\mathcal{C}$ using Hopcroft's algorithm again, yielding a DFA $\mathcal{D}$.

▶ **Lemma 13.** *The DFAs $\mathcal{A}$ and $\mathcal{D}$ of the above construction are almost equivalent.*

Proof. First, $\mathcal{A}$ and $\mathcal{B}$ are language equivalent.

To compare the language of $\mathcal{B}$ and $\mathcal{C}$, we note that, if p and q are almost equivalent, then so are $\delta(p, \sigma)$ and $\delta(q, \sigma)$ for all σ in Σ. (Assuming the opposite, there would be a word $\alpha \in \Sigma^\omega$ for which priority of the runs of $\mathcal{B}_{\delta(p,\sigma)}$ and $\mathcal{B}_{\delta(q,\sigma)}$ differ on infinitely many positions, which implied the same for $\sigma \cdot \alpha$ and runs on $\mathcal{B}_p$ and $\mathcal{B}_q$ and hence lead to a contradiction.)

Let us now consider runs $r_0^b r_1^b r_2^b r_3^b \dots$ and $r_0^c r_1^c r_2^c r_3^c \dots$ of $\mathcal{B}$ and $\mathcal{C}$ on some ω-word α. Then r_i^b and r_i^c are almost equivalent for all $i \in \omega$ by the above observation. Also, states in a run of $\mathcal{C}$ can never go down in the pre-order $(Q_+, \succeq)$. In particular, there is a bounded (at most $|Q|$) number of positions in the run, where $\mathcal{C}$ takes an adjusted transition—a transition $\delta'(q, \sigma) \neq \delta(q, \sigma)$—as this involves going strictly up in $(Q_+, \succeq)$. The number of positions $i \in \omega$ where either only r_i^b or only r_i^c are final can thus be estimated by the number of changed transitions taken times the bounded number of differences that can occur between almost equivalent states in $\mathcal{B}$.

Finally, $\mathcal{C}$ and $\mathcal{D}$ are again language equivalent. ◀

[2] This step is not necessary for the correctness of the algorithm or for its complexity.

An key observation for the proof that $\mathcal{D}$ is minimal is that almost equivalent states are in the same quotient class.

▶ **Lemma 14.** *Two states of $\mathcal{D}$ that are almost equivalent are in the same SCC.*

The proof that $\mathcal{D}$ is minimal builds on the fact that, whenever we go up in $(Q_+, \succeq)$, we choose the same representative.

▶ **Theorem 15.** *There is no DFA $\mathcal{E}$ almost equivalent to $\mathcal{D}$ that is strictly smaller than $\mathcal{D}$.*

Proof. For convenience, we now look at quotient classes of almost equivalent states that cover both $\mathcal{D}$ and $\mathcal{E}$ in this proof.

First, as $\mathcal{D}$ is minimal (among the language equivalent automata), all states in $\mathcal{D}$ are reachable. Let us assume that there is a smaller DFA $\mathcal{E}$ almost equivalent to $\mathcal{D}$. Then $\mathcal{E}$ must (at least) have the same quotient classes as $\mathcal{D}$, and hence, there must be a particular quotient class $[q]$ of $\mathcal{D}$ (and $\mathcal{E}$), such that there are strictly less representatives of this class in $\mathcal{E}$ than in $\mathcal{D}$.

By the previous lemma, the representatives of quotient classes of almost equivalent states of $\mathcal{D}$ are all in the same SCC. For trivial SCCs, this implies that there is only one representative in $\mathcal{D}$ and hence at least as many in $\mathcal{E}$.

For non-trivial SCCs, there is a witness of language non-equivalence that does not leave the SCC for all different occurrences. (Note that Construction 12 guarantees for $\mathcal{C}$ that, once an SCC is left, the target state—and hence the remainder of the run—is the same, no matter from which representative of a quotient class we start. And the proof of the previous lemma showed that the minimisation of $\mathcal{C}'$ is SCC preserving.)

As $\mathcal{E}$ has less representatives, we can pick one representative $r \in [q]$ of this class in $\mathcal{D}$ such that, for all representatives $e \in [q]$ in $\mathcal{E}$, we construct a finite word $\alpha_e \in \Sigma^*$ that is accepted either only by $\mathcal{E}_e$ or only by $\mathcal{D}_r$, such that the run of $\mathcal{D}_r$ on α_e stays in the SCC containing r. This invites a simple pumping argument: We can construct a word starting with a sequence β_0 that leads to r in $\mathcal{D}$. It also leads to some state e_1 almost equivalent to r in $\mathcal{E}$. Next, we continue our word with α_{e_1}, witnessing a difference. From the resulting state in $\mathcal{D}$, we continue with a non-empty sequence $\beta_1 \in \Sigma^+$ that brings us back to r. (We stay in the same SCC by construction.) Meanwhile, we have reached some state e_2 almost equivalent to r in $\mathcal{E}$. Next, we continue our word with α_{e_2}, witnessing a difference, and continue with a non-empty sequence $\beta_2 \in \Sigma^+$ that brings us back to r in $\mathcal{D}$, and so forth. We thus create an infinite sequence $\beta_0\alpha_{e_1}\beta_1\alpha_{e_2}\beta_2\alpha_{e_3}\ldots$ with infinitely many differences, which contradicts the almost equivalence of $\mathcal{D}$ and $\mathcal{E}$. ◀

▶ **Corollary 16.** *We can construct a minimal almost equivalent automaton to a given DFA $\mathcal{A}$ in time quadratic in the size of $\mathcal{A}$.* ◀

Note that the quadratic cost occurs only for constructing the quotients of almost equivalent states. Hence, there is a clear critical path, and improvement on this path would lead to an improvement of the overall algorithm.

It is interesting to observe that the minimal automaton almost equivalent to a weak automaton (when read as a DFA) obtained by Construction 12 is weak, and a language equivalent weak automata is almost equivalent. (An automaton is called weak if it recognises the same language when read as a DBA or as a DCA, or, similarly, if all states in the same SCC have the priority.)

▶ **Theorem 17.** *The algorithm from Construction 12 can be used to minimise weak automata.*

Almost equivalence can hence be read as a generalisation of language equivalence of weak automata.

4.3 Space Reduction for DBAs and DCAs

The techniques introduced for finding minimal almost equivalent automata can easily be adjusted to stronger state-space reductions for DBAs and DCAs: If we use language equivalence for the respective automata instead of almost equivalence, the resulting automaton remains language equivalent.

▶ **Theorem 18.** *Swapping quotients of almost equivalent states for the coarser quotients of language equivalent states for DBAs and DCAs in Construction 12 provides a language equivalent automaton $\mathcal{D}$, and the cost remains quadratic in the size of $\mathcal{A}$.*

An interesting corollary from the proofs (see [21]) of Theorems 18 and 15 is:

▶ **Corollary 19.** *Minimisation techniques for DBAs or DCAs can treat the individual SCCs of the resulting automaton $\mathcal{D}$ individually.* ◀

An interesting aspect of this minimisation is that we can treat a local version of weak automata: We call an SCC weak if all infinite paths within this SCC are accepting or all infinite paths within this SCC are rejecting. For weak SCCs, we can obviously make all states accepting or rejecting, respectively, without changing the language of a DBA or DCA.

Doing so in the automaton $\mathcal{C}$ in from Construction 12 leads to all states equivalent by the respective equivalence relation (almost equivalence or language equivalence as DBA or DCA) becoming language equivalent when the automaton is read as a DFA, and are therefore merged in $\mathcal{D}$. Thus, there is exactly one of these states in $\mathcal{D}$, and the $\mathcal{D}$ is locally optimal.

A further tractable minimisation would be to greedily merge states: For an automaton $\mathcal{A}$ we denote with $\mathcal{A}^{p \triangleright q}$ the automaton that results from changing the transition function δ to δ' such that $\delta'(r, \sigma) = q$ if $\delta(r, \sigma) = p$ and $\delta'(r, \sigma) = \delta(r, \sigma)$ otherwise, choosing q as initial state if q was the former initial state, and removing p from the state-space. A natural tractable minimisation would be to greedily consider $\mathcal{A}^{p \triangleright q}$ for language equivalent states p and q until no further states can be merged. Note that, by Corollary 19, it suffices to look at the respective SCCs only, which may speed up the computation significantly.

This is even more important for exhaustive search for minimal automata, such as the SAT based methods suggested by Ehlers [5].

5 Discussion

This paper has two main results: First, it establishes that minimising deterministic Büchi, Co-Büchi and parity automata are NP-complete problems.

A second central contribution is the introduction of relative minimisation of DFAs, a powerful technique to minimise deterministic finite automata when allowing for minor differences in their language. This natural minimisation problem on DFAs is strictly between the problem of hyper-minimising DFAs and minimising DBAs or DCAs and can be viewed as a generalisation of the minimisation problem of weak automata. We show that the relative minimisation of DFAs is tractable and provide a simple quadratic algorithm.

Finally, we strengthened this algorithm by relaxing the requirement for merging states from almost to language equivalent states, which provides a promising technique to reduce the state-space of DBAs and DCAs. This technique does not only have the potential to reduce the state-space of the automaton significantly, it also suffices to focus on its SCCs when seeking to reduce the state-space of the automaton further. This can be used to accelerate further reduction heuristics—like the greedy merge discussed—and exhaustive search methods alike.

References

1. Andrew Badr. Hyper-minimization in $O(n^2)$. *International Journal of Foundations of Computer Science*, 20(4):735–746, 2009.
2. Andrew Badr, Viliam Geffert, and Ian Shipman. Hyper-minimizing minimized deterministic finite state automata. *Informatique Théorique et Applications*, 43(1):69–94, 2009.
3. Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*. MIT Press, 2008.
4. Olivier Carton and Ramón Maceiras. Computing the rabin index of a parity automaton. *Theoretical Informatics and Applications (ITA)*, 33(6):495–506, 1999.
5. Rüdiger Ehlers. Minimising deterministic Büchi automata precisely using SAT. In *Proc. of SAT*, pages 326–332, 2010.
6. Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Proc. of LICS*, pages 321–330, 2005.
7. Paweł Gawrychowski and Artur Jeż. Hyper-minimisation made efficient. In *In Proc. of MFCS*, pages 356–368, 2009.
8. John E. Hopcroft. An n log n algorithm for minimizing states in a finite automaton. Technical Report CS-190, 1970.
9. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2000.
10. Neil Immerman. Nondeterministic space is closed under complementation. *SIAM Journal on Computing*, 17(5):935–938, 1988.
11. Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *Proc. of LICS*, pages 389–398, 2001.
12. Christoph Löding. Efficient minimisation of deterministic weak automata. *Information Processing Letters*, 79(3):105–109, 2001.
13. P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *Proc. of ICALP*, pages 396–407, 2001.
14. Damian Niwinski and Igor Walukiewicz. Relating hierarchies of word and tree automata. In *Proc. of STACS*, pages 320–331. 1998.
15. Nir Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. *Journal of Logical Methods in Computer Science*, 3(3:5), 2007.
16. Amir Pnueli and Roni Rosner. On the synthesis of a reactive module. In *Proc. of POPL*, pages 179–190, 1989.
17. Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *Proc. of ICALP*, pages 652–671, 1989.
18. Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Proc. FOCS*, pages 746–757. 1990.
19. Michael O. Rabin. *Automata on Infinite Objects and Church's Problem*, volume 13 of *Regional Conference Series in Mathematics*. American Mathematical Society, 1972.
20. Sven Schewe. Tighter bounds for the determinisation of Büchi automata. In *Proc. of FoSSaCS*, pages 167–181, 2009.
21. Sven Schewe. Minimisation of deterministic parity and buchi automata and relative minimisation of deterministic finite automata. *CoRR*, abs/1007.1333, 2010.
22. Sven Schewe and Bernd Finkbeiner. Synthesis of asynchronous systems. In *Proc. of LOPSTR*, pages 127–142, 2006.
23. Igor Walukiewicz and Swarup Mohalik. Distributed games. In *Proc. of FSTTCS*, pages 338–351, 2003.

Parityizing Rabin and Streett*

Udi Boker[1], Orna Kupferman[1], and Avital Steinitz[1]

1 **Hebrew University of Jerusalem, School of Engineering and Computer Science**
{udiboker,orna,avinu}@cs.huji.ac.il

Abstract

The parity acceptance condition for ω-regular languages is a special case of the Rabin and Streett acceptance conditions. While the parity acceptance condition is as expressive as the richer conditions, in both the deterministic and nondeterministic settings, Rabin and Streett automata are more succinct, and their translation to parity automata may blow-up the state space. The appealing properties of the parity condition, mainly the fact it is dualizable and allows for memoryless strategies, make such a translation useful in various decision procedures.

In this paper we study languages that are recognizable by an automaton on top of which one can define both a Rabin and a Streett condition for the language. We show that if the underlying automaton is deterministic, then we can define on top of it also a parity condition for the language. We also show that this relation does not hold in the nondeterministic setting. Finally, we use the construction of the parity condition in the deterministic case in order to solve the problem of deciding whether a given Rabin or Streett automaton has an equivalent parity automaton on the same structure, and show that it is PTIME-complete in the deterministic setting and is PSPACE-complete in the nondeterministic setting.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.412

1 Introduction

Finite *automata on infinite objects* are widely used for the specification, verification, and synthesis of nonterminating systems [3, 15, 21]. Since a run of an automaton on an infinite word does not have a final state, acceptance is determined with respect to the set of states visited infinitely often during the run. There are many ways to classify an automaton on infinite words. One is the class of its acceptance condition. For example, in *Büchi* automata, some of the states are designated as accepting states, and a run is accepting iff it visits states from the accepting set infinitely often [1]. More general are *Rabin* automata. Here, the acceptance condition is a set $\alpha = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$ of pairs of sets of states, and a run is accepting if there is a pair $\langle G_i, B_i\rangle$ for which the set of states visited infinitely often intersects G_i and does not intersect B_i. The condition α can also be viewed as a *Streett* condition, in which case a run is accepting if for all pairs $\langle G_i, B_i\rangle$, if the set of states visited infinitely often intersects G_i, then it also intersects B_i. Note that the Rabin and Streett conditions dualize each other. Thus, a run satisfies α when viewed as a Rabin condition iff it does not satisfy α when viewed as a Streett condition. The analysis of logics with fixed-points led to extensive study of the *parity* acceptance condition [5, 17]. There, the acceptance condition is a sequence $\{F_1, F_2, \ldots, F_{2k}\}$ of sets of states, and a run is accepting iff the minimal index i for which the set F_i is visited infinitely often is even. It is not hard to see that the parity condition is a special case of both the Rabin and Streett conditions, in the sense that a given parity condition can be translated to equivalent Rabin and Streett conditions. The number of pairs or sets in the acceptance conditions is referred to as the

* The first author is supported in part by a Lady Davis postdoctoral fellowship.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 412–423

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

index of the automaton. We use NRW, NSW, and NPW to denote nondeterministic Rabin, Streett, and parity word automata, respectively, and use DRW, DSW, and DPW to denote the corresponding deterministic automata. We sometimes add the number of states and index. So, for example, NRW(n, k) is a nondeterministic automaton with n states and index k.

The type of an automaton influences its succinctness. For example, while Rabin, Streett, and parity automata all recognize all ω-regular languages, the translation of a DRW to a DSW (or vise versa) may involve a blow-up exponential in the index, and so does the translation of a DRW or a DSW to a DPW [16]. The succinctness of Rabin and Streett automata with respect to parity automata is carried over to the nondeterministic setting [19, 20]. The type of an automaton also influences the difficulty of constructions and decision problems for it. For example, while complementation of DPWs is straightforward, as it is easy to dualize a parity condition, complementation of DRWs and DSWs involves a translation of the dual DSWs and DRWs, respectively, back to DRWs and DSWs, which, as described above, involves an exponential blow-up. As another example, while the nonemptiness problem for DPW(n, k) can be solved in time $O(n \log k)$ [10] and is NLOGSPACE-complete, the one for DRW(n, k) is still NLOGSPACE-complete but needs time $O(nk)$, whereas the one for DSW(n, k) is PTIME-complete [6], with on-going research on the precise, larger than $O(nk)$, bound [6, 8]. Thus, the succinctness of Rabin and Streett automata is traded-off by more complex constructions and algorithms. Finally, only the parity acceptance condition allows for memoryless strategies for both players [5]. The fact parity games are memoryless is of great importance in synthesis algorithms, where one wants to generate transducers for the winning strategies [4]. The fact both players have memoryless strategies is useful in settings in which one considers strategies for both the system and its environment [13]. A good evidence to the superiority of the parity condition in the application front is the fact that the highlight of Piterman's determinization construction for nondeterministic Büchi automata [18] has been the fact it generates a DPW, rather than the DRW generated by Safra's construction [19], and less the saving in the state space it suggests.

Recall that while the parity condition can be translated to the Rabin and Streett conditions, the other direction is not valid: the translations of Rabin and Streett automata to parity automata cannot only modify the acceptance condition and they involve automata with different, and substantially bigger, state spaces. In some cases, it is possible to translate automata with a particular acceptance condition to automata with a weaker acceptance condition without modifying the state space. For example, it is shown in [11] that DRWs are *Büchi type*: if a DRW has an equivalent deterministic Büchi automaton, then there is also an equivalent deterministic Büchi automaton on top of the same structure. Additional examples of typeness for ω-regular languages are studied in [12]. We would like to study typeness for parity automata, and in particular the ability to modify Rabin and Streett conditions to an equivalent parity condition.

The connection between the combination of Rabin and Streett with the parity condition was studied in the context of *two-player games* in [22]. There, Zielonka shows that if the winning condition of a finitely colored game can be specified as both Rabin and Streett conditions, then it can also be characterized by a parity (or chain, as it is called there) condition. In this paper we study this connection in the context of automata on infinite words: Suppose that some language can be defined on top of the same automaton by both Rabin and Streett conditions. Can we define an equivalent parity condition on top of the same automaton? Before describing our results, let us mention that they do not follow directly from Zielonka's result. In fact, our results are part of a general effort of lifting results from

the world of two-player games to the world of automata. By [7], the nonemptiness problem for nondeterministic tree automata can be reduced to the solution of a two-player game. The connection between games and automata is further formalized in [14]. As shown there, since transitions of games are not associated with letters, games correspond to alternating word automata over a singleton alphabet, and one cannot talk about a language of a game. Indeed, results and methods that hold for games cannot in general be applied to automata. For example, while today there are several algorithms that solve parity games in time less than $O(n^k)$ [9], the best translation of alternating parity word automata to alternating weak word automata (for which the 1-letter nonemptiness problem can be solved in linear time) involves an $O(n^k)$ blow up, where n is the number of states and k is the index of the parity condition. The challenge has to do with the fact that reasoning about games one can abstract components of the game, whereas translations among automata must keep the exact same language – every letter counts.

Back to our problem, our main result states that if the automaton is deterministic, then one can define an equivalent parity condition on top of it! Formally, if $\mathcal{A}$ is a deterministic automaton with n states and there is a Rabin condition α of index k and a Streett condition β of index l such that the language of $\mathcal{A}$ with α is equal to the language of $\mathcal{A}$ with β, then there exists a parity condition γ of index at most $\min\{2k+2, 2l+2, n+2\}$ such that the language of $\mathcal{A}$ with γ is equal to the language of $\mathcal{A}$ with α and β. Our proof is constructive, it proceeds by induction on the index of the constructed parity automaton, and it involves a decomposition of $\mathcal{A}$ to its maximal strongly connected components, applications of the translation on them, and a composition of the underlying parity conditions to a global one. We study also the nondeterministic setting and show that the determinism of $\mathcal{A}$ is essential. That is, we show that there is a nondeterministic automaton $\mathcal{A}$ such that there are Rabin and Streett conditions on top of $\mathcal{A}$ that define the same language, and still no parity automaton for the language can be defined on top of $\mathcal{A}$. This result is another evidence to the importance of the alphabet and the fact the setting of automata is different than the one of games studied in [22]. Indeed, every nondeterministic automaton can be made deterministic by enriching its alphabet (c.f., the *cylindrification* techniques of [2]).

In addition to formalizing the intuition of "parity is the intersection of Rabin and Streett" and introducing a blow-up-free translation to DPW, a careful analysis of the construction of the equivalent parity condition shows that it is independent of the Streett condition and relies only on its existence. Consequently, we can use the construction in order to decide whether a given DRW can be translated to an equivalent DPW on the same structure. We show that this problem is PTIME-complete. Note that the duality between the Rabin and Streett conditions and the self-duality of the parity condition imply that the problem of deciding whether a given DSW can be translated to a DPW on the same structure is PTIME-complete too. In addition, we prove that the problem of deciding whether a given NRW or NSW has an equivalent NPW on the same structure is PSPACE-complete.

2 Preliminaries

Automata on infinite words. Given an alphabet Σ, an *infinite word over* Σ is an infinite sequence $w = \sigma_0 \cdot \sigma_1 \cdot \sigma_2 \cdots$ of letters in Σ. We denote by w^l the suffix $\sigma_l \cdot \sigma_{l+1} \cdot \sigma_{l+2} \cdots$ of w. An *automaton on infinite words* is $\mathcal{U} = \langle \Sigma, Q, \delta, Q_{in}, \alpha \rangle$, where Σ is the input alphabet, Q is a finite set of states, $\delta : Q \times \Sigma \to 2^Q$ is a transition function, $Q_{in} \subseteq Q$ is a set of initial states, and α is an acceptance condition (a condition that defines a subset of Q^ω).

Since the transition function of $\mathcal{U}$ may specify many possible transitions for each state and letter and since the initial state may be one of the possibly few states in Q_{in}, $\mathcal{U}$ is not

deterministic. If δ is such that for every $q \in Q$ and $\sigma \in \Sigma$, we have that $|\delta(q,\sigma)| = 1$ and if $|Q_{in}| = 1$, then $\mathcal{U}$ is a deterministic automaton. When $\mathcal{U}$ is deterministic we refer to the single state in Q_{in} by q_{in} and to δ as to a function from Σ^* to Q (rather than to 2^Q). We sometimes refer to the transition function δ of a deterministic automaton as a function $\delta : \Sigma^* \to Q$, where $\delta(\epsilon) = q_{in}$ and $\delta(w \cdot \sigma) = \delta(\delta(w), \sigma))$. Thus $\delta(w)$ is the state that $\mathcal{U}$ visits after reading w. We say that a state $q \in Q$ is reachable in $\mathcal{U}$ if there is a finite word w such that $\delta(w) = q$.

A *run* of $\mathcal{U}$ on w is an infinite word $r = q_0 \cdot q_1 \cdot q_2 \cdots$ over Q, where $q_0 \in Q_{in}$ (i.e., the run starts in an initial state) and for every $l \geq 0$, we have $q_{l+1} \in \delta(q_l, \sigma_l)$ (i.e., the run obeys the transition function). In automata over finite words, acceptance is defined according to the last state visited by the run. When the words are infinite, there is no such thing "last state", and acceptance is defined according to the set $inf(r)$ of states that r visits *infinitely often*, i.e., $inf(r) = \{q \in Q : \text{for infinitely many } l \in \mathbb{N}, \text{we have } r_l = q\}$. Hence, acceptance is *prefix independent*, i.e. for all runs r_1, r_2 such that $r_1^l = r_2^m$ for some l and m we have that r_1 is accepting iff r_2 is accepting. As Q is finite, it is guaranteed that $inf(r) \neq \emptyset$. A run r is accepting iff the set $inf(r)$ satisfies the acceptance condition of $\mathcal{U}$. Several acceptance conditions are studied in the literature. We consider here three:

- *Rabin automata*, where $\alpha = \{\langle G_1, B_1\rangle, \langle G_2, B_2\rangle, \ldots, \langle G_k, B_k\rangle\}$, and $inf(r)$ satisfies α iff for some $1 \leq i \leq k$, we have that $inf(r) \cap G_i \neq \emptyset$ and $inf(r) \cap B_i = \emptyset$.
- *Streett automata*, where $\alpha = \{\langle L_1, U_1\rangle, \langle L_2, L_2\rangle, \ldots, \langle L_l, U_l\rangle\}$, and $inf(r)$ satisfies α iff for all $1 \leq i \leq l$, if $inf(r) \cap L_i \neq \emptyset$, then $inf(r) \cap U_i \neq \emptyset$.
- *parity automata*, where $\alpha = \{F_1, F_2, \ldots, F_{2k}\}$ with $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_{2k} = Q$, and $inf(r)$ satisfies α iff the minimal index i for which $inf(r) \cap F_i \neq \emptyset$ is even.

The number of sets in the parity acceptance condition or pairs in the Rabin and Streett acceptance conditions is called the *index* of α (or $\mathcal{U}$). Note that the Rabin and Streett conditions are dual, in the sense that a set S satisfies a Rabin condition α iff S does not satisfy α when viewed as a Streett condition. Similarly, the parity condition is dual to itself, in the sense that a set S satisfies a parity condition $\{F_1, F_2, \ldots, F_{2k}\}$ iff S does not satisfy the parity condition $\{\emptyset, F_1, F_2, \ldots, F_{2k}, F_{2k}\}$.

Since $\mathcal{U}$ may not be deterministic, it may have many runs on w. In contrast, a deterministic automaton has a single run on w. An automaton $\mathcal{U}$ is said to accept an input word w iff there exists an accepting run of $\mathcal{U}$ on w. This implies that if $\mathcal{U}$ is deterministic it accepts an input word w iff the single run of $\mathcal{U}$ on w is accepting. The *language* of $\mathcal{U}$, denoted $\mathcal{L}(\mathcal{U})$, is the set of words $\mathcal{U}$ accepts.

A (deterministic) *pre-automaton* $\mathcal{A} = \langle \Sigma, Q, \delta, Q_{in}\rangle$ is a (deterministic, respectively) automaton with no acceptance condition. For an acceptance condition α we use $\mathcal{L}(\mathcal{A}, \alpha)$ to denote the language of the automaton $\mathcal{U} = \langle \mathcal{A}, \alpha\rangle$.

For a pre-automaton $\mathcal{A}$ and a state $q \in Q$, let $\mathcal{A}^q$ denote the pre-automaton $\langle \Sigma, Q, \delta, \{q\}\rangle$. That is, $\mathcal{A}^q$ is the pre-automaton $\mathcal{A}$ except for having q as its single initial state (we sometimes abuse notations and omit the { } around q). For a pre-automaton $\mathcal{A}$, a subset $C \subseteq Q$ and a state $q \in C$, let $\mathcal{A}|_C^q$ denote the pre-automaton $\langle \Sigma, C, \delta|_C, q\rangle$ where $\delta|_C$ is the restrictions of δ to C, i.e. $\delta|_C : C \times \Sigma \to 2^C$ is such that $\delta|_C(q,\sigma) = \delta(q,\sigma) \cap C$. For an acceptance condition α, denote by $\alpha|_C$ the condition that is obtained from α by intersecting all its sets with C.

The *underlying graph* of a pre-automaton $\mathcal{A}$, denoted $G_{\mathcal{A}}$, is the graph $\langle Q, E\rangle$, where $E(q,q')$ iff there is a letter $\sigma \in \Sigma$ such that $q' \in \delta(q,\sigma)$. A *strongly connected component* of a graph $G = \langle Q, E\rangle$ is a set of vertices $C \subseteq Q$ such that every two states $q, q' \in C$ are reachable from each other. A *maximal strongly connected component* (MSCC) in a graph G is a strongly connected component C such that for all nonempty sets of vertices $C' \in G \setminus C$ the

set $C \cup C'$ is not strongly connected. A graph is said to be strongly connected if its vertices consist a single strongly connected component. A pre-automaton is said to be strongly connected if its underlying graph is strongly connected.

2.1 Simple Translations

Syntactic Translations. Parity automata can be viewed as a special case of Rabin and of Streett automata. It is easy to see that a parity condition $\{F_1, F_2, \ldots, F_{2k}\}$ is equivalent to the Streett condition $\{\langle F_{2k-1}, F_{2k-2}\rangle, \ldots, \langle F_3, F_2\rangle, \langle F_1, \emptyset\rangle\}$ and to the Rabin condition $\{\langle F_{2k}, F_{2k-1}\rangle, \ldots, \langle F_4, F_3\rangle, \langle F_2, F_1\rangle\}$. Similarly, a Rabin condition with a single pair $\langle G, B\rangle$, is equivalent to the parity condition $\{B, B \cup G, Q, Q\}$. Generalizing, it is not hard to see that a Rabin condition $\alpha = \{\langle G_1, B_1\rangle, \ldots, \langle G_k, B_k\rangle\}$ with nested "bad" sets, i.e. $B_1 \subseteq B_2 \subseteq \ldots \subseteq B_k$, is equivalent to the parity condition $\gamma = \{B_1, B_1 \cup G_1, B_1 \cup G_1 \cup B_2, B_1 \cup G_1 \cup B_2 \cup G_2, \ldots\}$.

"Typed" Translations. "Typed" translations depend on the structure of the associated pre-automaton. For instance, according to [11], deterministic Rabin automata are Büchi type. That is, given a pre-automaton $\mathcal{A}$ and a Rabin condition α, whenever $\mathcal{L}(\mathcal{A}, \alpha)$ is Büchi recognizable there exists a Büchi condition $\beta \subseteq Q$, which is equivalent to the parity condition $\{\emptyset, \beta, Q, Q\}$, such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \beta)$. However, the computation of β requires an examination of $\mathcal{A}$ and does not depend only on the syntax of α. As another example, the nested "bad" sets condition above can be relaxed to reproduce another typed translation between Rabin and parity conditions. Indeed, if we replace the $B_i \subseteq B_{i+1}$ condition by a weaker one, namely that for every cycle w in $G_{\mathcal{A}}$, it holds that $w \cap B_i \neq \emptyset \Rightarrow w \cap B_{i+1} \neq \emptyset$, then the translation is still valid, does not change the structure of the automaton, but depends on it. Our goal is to extend the applicability of typed translations to DPWs.

3 The Deterministic Case

In this section we show that a DRW has an equivalent DPW on the same pre-automaton iff it has an equivalent DSW on it. Obviously, the dual result holds when starting from a DSW. Our proof is constructive, providing a polynomial procedure for generating the equivalent parity condition or returning the answer that such a condition does not exist.

Our proof is iterative, proceeding by induction on the index of the generated parity automaton. The first iteration is simple – all the states that cannot be visited infinitely often in a Rabin accepting run (defined below as the "hopeless" states) are gathered to the first (odd) set of the parity condition. After each iteration, the states gathered so far are removed from the pre-automaton, which is then decomposed into maximally strongly connected components. The next iterations are done separately for each component. The second iteration looks for a Rabin pair of the form $\langle G_i, \emptyset\rangle$ (having no "bad" states). If such a pair exists, then its "ultimately good" states are gathered to the next (even) set of the parity condition. The procedure continues, gathering the hopeless states in odd iterations and the ultimately good states in even iterations. In the end, the parity conditions for the separated components are composed to a global condition. The main observation is that an equivalent Streett condition guarantees the existence of the required $\langle G_i, \emptyset\rangle$ pair in every iteration.

We start with defining the notion of "hopeless states". Consider a pre-automaton $\mathcal{A}$ and a Rabin condition α over Q. A state q of $\mathcal{A}$ is *hopeless in $\mathcal{A}$ with respect to α* iff every run r of $\mathcal{A}$ that visits q infinitely often is rejecting. Thus, for every run r of $\mathcal{A}$, if $q \in inf(r)$ then r is rejecting with respect to α. Let $\mathcal{H}_{\mathcal{A},\alpha}$ denote the set of all the states that are hopeless

in $\mathcal{A}$ with respect to α. We say that a pre-automaton $\mathcal{A}$ is *hopeless-free* with respect to a condition α iff $\mathcal{H}_{\mathcal{A},\alpha} = \emptyset$.

Note that once a state is hopeless with respect to some condition, it is hopeless for all other equivalent conditions on the same deterministic pre-automaton. We formalize this in the following lemma.

▶ **Lemma 1.** *Let $\mathcal{A}$ be a deterministic pre-automaton and α and β two acceptance conditions over Q such that $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\beta)$. Then, $\mathcal{H}_{\mathcal{A},\alpha} = \mathcal{H}_{\mathcal{A},\beta}$.*

Proof. Once we show that $\mathcal{H}_{\mathcal{A},\alpha} \subseteq \mathcal{H}_{\mathcal{A},\beta}$ the lemma will follow from symmetry. If both $\mathcal{H}_{\mathcal{A},\alpha}$ and $\mathcal{H}_{\mathcal{A},\beta}$ are empty then they are clearly equal. Otherwise, assume w.l.o.g. that $\mathcal{H}_{\mathcal{A},\alpha} \neq \emptyset$ and consider a state $q \in \mathcal{H}_{\mathcal{A},\alpha}$. If no run of $\mathcal{A}$ visits q infinitely often, then $q \in \mathcal{H}_{\mathcal{A},\beta}$ vacuously. Otherwise, consider a run r of $\mathcal{A}$ such that $q \in \mathit{inf}(r)$. Consider a word w over Σ such that r is a run of $\mathcal{A}$ on w. Since $q \in \mathcal{H}_{\mathcal{A},\alpha}$ r is rejecting with respect to α and $w \notin \mathcal{L}(\mathcal{A},\alpha)$. Since $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\beta)$ we have $w \notin \mathcal{L}(\mathcal{A},\beta)$. Since $\mathcal{A}$ is deterministic r is the single run of $\mathcal{A}$ on w, therefore r is also rejecting with respect to β.

We therefore showed that for every r of $\mathcal{A}$ if $q \in \mathit{inf}(r)$ then r is rejecting with respect to β, therefore $q \in \mathcal{H}_{\mathcal{A},\beta}$. ◀

The next lemmas justify our decomposition and re-composition steps, showing that the equivalence of acceptance conditions is carried over to and from hopeless-free MSCCs.

▶ **Lemma 2** (Zoom In). *Let $\mathcal{A} = \langle \Sigma, Q, \delta, Q_{in} \rangle$ be a deterministic pre-automaton and let α and β be two prefix-independent acceptance conditions such that $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\beta)$. Then, for every $C \subseteq Q$ and reachable state $q \in C$ we have $\mathcal{L}(\mathcal{A}|_C^q, \alpha|_C) = \mathcal{L}(\mathcal{A}|_C^q, \beta|_C)$.*

Proof. From symmetry it suffices to show that $\mathcal{L}(\mathcal{A}|_C^q, \alpha|_C) \subseteq \mathcal{L}(\mathcal{A}|_C^q, \beta|_C)$. Consider a word $w \in \mathcal{L}(\mathcal{A}|_C^q, \alpha|_C)$, and let r be the run of $\mathcal{A}|_C^q$ on w (r is well defined since $\mathcal{A}$ is deterministic, thus so is $\mathcal{A}|_C^q$). Since r is accepting with respect to $\alpha|_C$, the set $\mathit{inf}(r)$ satisfies the condition $\alpha|_C$.

Since $\mathit{inf}(r) \subseteq C$, for every set $S \subseteq Q$ we have that $\mathit{inf}(r) \cap (S \cap C) = \emptyset$ iff $\mathit{inf}(r) \cap S = \emptyset$. Since $\alpha|_C$ is obtained from α by intersecting its sets with C, it follows that $\mathit{inf}(r)$ satisfies also α, hence $w \in \mathcal{L}(\mathcal{A}^q, \alpha)$.

Consider a word $v \in \Sigma^*$ such that $\delta(v) = q$. Such a word clearly exists because q is reachable. Let r' be the run of $\mathcal{A}$ on $v \cdot w$ (the concatenation of the two words). Since $\mathit{inf}(r') = \mathit{inf}(r)$ we have that r' is an accepting run of $\mathcal{A}$ with respect to α, thus $v \cdot w \in \mathcal{L}(\mathcal{A},\alpha)$. Therefore, we have $v \cdot w \in \mathcal{L}(\mathcal{A},\beta)$, and since $\mathcal{A}$ is deterministic it follows that r' is accepting with respect to β. Again, since $\mathit{inf}(r') = \mathit{inf}(r)$, we get that r is an accepting run of $\mathcal{A}^q$ with respect to β, and since $\mathit{inf}(r) \subseteq C$, we get that r is an accepting run of $\mathcal{A}|_C^q$ with respect to $\beta|_C$. Thus, $w \in \mathcal{L}(\mathcal{A}|_C^q, \beta|_C)$ as required. ◀

▶ **Lemma 3** (Zoom Out). *Let $\langle \mathcal{A}, \alpha \rangle$ be a deterministic hopeless-free Rabin automaton. Let $\mathcal{C}$ be the set of MSCCs of $G_{\mathcal{A}}$. For every MSCC $C \in \mathcal{C}$, let γ_C be a parity condition of index m_C such that for every state $q \in C$, it holds that $\mathcal{L}(\mathcal{A}|_C^q, \alpha|_C) = \mathcal{L}(\mathcal{A}|_C^q, \gamma_C)$. Then, there is a parity condition γ of index $\max_{C \in \mathcal{C}} m_C$ such that $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\gamma)$.*

Proof. For every $C \in \mathcal{C}$, let $\gamma_C = \{F_{C,1}, F_{C,2}, \ldots, F_{C,m_C}\}$, and let $m_{\mathcal{C}} = \max_{C \in \mathcal{C}} m_C$. We extend all γ_C to be of index $m_{\mathcal{C}}$ (this can be done by padding the condition with C sets). We define $\gamma = \{F_1, F_2, \ldots, F_{m_{\mathcal{C}}}\}$, where for all $1 \leqslant i \leqslant m_{\mathcal{C}}$, we have $F_i = \bigcup_{C \in \mathcal{C}} F_{C,i}$. We prove that $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\gamma)$.

We first prove that $\mathcal{L}(\mathcal{A},\alpha) \subseteq \mathcal{L}(\mathcal{A},\gamma)$. Consider a word $w \in \mathcal{L}(\mathcal{A},\alpha)$. Let r be the run of $\mathcal{A}$ on w. Then, $\mathit{inf}(r)$ satisfies α. Also, since r is an infinite path in $G_{\mathcal{A}}$ there is a single

$C \in \mathcal{C}$ such that $inf(r) \subseteq C$, thus in particular $inf(r) \cap F_{C',i} = \emptyset$ for every $C' \in \mathcal{C}$ other than C and for every $1 \leqslant i \leqslant m_{\mathcal{C}}$. Thus, the minimal index i for which $inf(r) \cap F_i \neq \emptyset$ is equal to the minimal index i' for which $inf(r) \cap F_{C,i'} \neq \emptyset$.

To see that i' is even, note that there exists an index $l \geqslant 0$ such that $r^l \subseteq C$ and denote $r_l = q$. Because $inf(r) = inf(r^l)$ we have that $w^l \in \mathcal{L}(\mathcal{A}|_C^q, \alpha|_C)$ and therefore also $w^l \in \mathcal{L}(\mathcal{A}|_C^q, \gamma_C)$, thus i' is even.

The other direction is similar. ◀

In fact, Lemma 3 above is valid also for automata that are not hopeless-free. To see this, note that it would be enough to define the sets in γ as $F_i = \mathcal{H}_{\mathcal{A},\alpha} \cup \bigcup_{C \in \mathcal{C}} F_{C,i}$, thus forbidding any accepting run that would be accepting according to γ from visiting hopeless states infinitely often, and therefore restricting the accepting runs to the same MSCCs. We therefore have the following:

▶ **Lemma 4** (Zoom Out). *Let $\langle \mathcal{A}, \alpha \rangle$ be a deterministic Rabin automaton. Let $\mathcal{C}$ be the set of MSCCs of $G_{\mathcal{A}'}$, where $\mathcal{A}'$ is the restriction of $\mathcal{A}$ to its non-hopeless states. For every MSCC $C \in \mathcal{C}$, let γ_C be a parity condition of index m_C such that for every state $q \in C$, it holds that $\mathcal{L}(\mathcal{A}|_C^q, \alpha|_C) = \mathcal{L}(\mathcal{A}|_C^q, \gamma_C)$. Then, there is a parity condition γ of index $\max_{C \in \mathcal{C}} m_C$ such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \gamma)$.*

The Streett Limitation. Lemmas 2 and 4 suggest that we restrict our attention to deterministic strongly connected Rabin and Streett automata that are hopeless-free. The Lemma below provides the key observation that under these conditions one of the "bad" Rabin sets must be empty.

▶ **Lemma 5.** *Let $\mathcal{A}$ be a strongly connected deterministic pre-automaton, and let $\alpha = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$ and $\beta = \{\langle L_1, U_1 \rangle, \ldots, \langle L_l, U_l \rangle\}$ be Rabin and Streett conditions such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \beta)$. Further assume that $\mathcal{A}$ is hopeless-free with respect to the equivalent conditions α and β. Then, there must be an index $1 \leqslant i \leqslant k$ for which $B_i = \emptyset$.*

Proof. Consider first the Streett condition β. Assume that there is an index $1 \leqslant j \leqslant l$ such that $U_j = \emptyset$. Then, all the states in L_j are hopeless with respect to β. Since $\mathcal{A}$ is hopeless-free with respect to β it follows that if $U_j = \emptyset$ then $L_j = \emptyset$, thus the pair $\langle U_j, L_j \rangle$ can be removed from β. Therefore we can assume that for all $1 \leqslant j \leqslant l$, the set U_j is not empty. Now, consider the Rabin condition α and assume by way of contradiction that for all $1 \leqslant i \leqslant k$ we have $B_i \neq \emptyset$. Consider a word w such that the run r over w visits infinitely often all the states of $\mathcal{A}$. Such a word clearly exists, because $\mathcal{A}$ is strongly connected. Since r visits all the sets B_i of α infinitely often it does not satisfy α. Thus $w \notin \mathcal{L}(\mathcal{A}, \alpha)$. On the other hand, since r visits all the sets U_j (non of which is empty) infinitely often, it does satisfy β, thus $w \in \mathcal{L}(\mathcal{A}, \beta)$, contradicting the equivalence of $\mathcal{L}(\mathcal{A}, \alpha)$ and $\mathcal{L}(\mathcal{A}, \beta)$ ◀

We continue to the main lemma, proving a special case of the desired theorem.

▶ **Lemma 6.** *Let $\mathcal{A}$ be strongly connected deterministic pre-automaton, and let $\alpha = \{\langle G_1, B_1 \rangle, \ldots, \langle G_k, B_k \rangle\}$ and $\beta = \{\langle L_1, U_1 \rangle, \ldots, \langle L_l, U_l \rangle\}$ be Rabin and Streett conditions such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \beta)$. Further assume that $\mathcal{A}$ is hopeless-free with respect to the equivalent conditions α and β. Then, there is a parity acceptance condition γ of index at most $2k + 2$ such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \beta) = \mathcal{L}(\mathcal{A}, \gamma)$.*

Proof. The proof proceeds by induction on the index of the Rabin condition. When $k = 1$ it is easy, as $\alpha = \langle G, B \rangle$ is equivalent to the parity condition $\{B, B \cup G, Q, Q\}$. We

assume by induction that the claim holds for Rabin conditions of index at most $k-1$. Formally, we assume that given a strongly connected deterministic pre-automaton $\mathcal{A}'$, Rabin and Streett conditions $\alpha = \{\langle G_1, B_1\rangle, \ldots, \langle G_{k-1}, B_{k-1}\rangle\}$ and $\beta = \{\langle L_1, U_1\rangle, \ldots, \langle L_{l'}, U_{l'}\rangle\}$ such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \beta)$ and such that $\mathcal{A}'$ is hopeless-free with respect to them, we know how to construct a parity acceptance condition γ of index at most $2k$, such that $\mathcal{L}(\mathcal{A}, \alpha') = \mathcal{L}(\mathcal{A}, \beta') = \mathcal{L}(\mathcal{A}, \gamma')$.

We consider a Rabin condition α of index k and decompose it into two Rabin conditions, α' and α'', such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \alpha') \cup \mathcal{L}(\mathcal{A}, \alpha'')$. Next, using the induction hypothesis we will construct parity conditions γ' and γ'' such that $\mathcal{L}(\mathcal{A}', \alpha') = \mathcal{L}(\mathcal{A}', \gamma')$ and $\mathcal{L}(\mathcal{A}', \alpha'') = \mathcal{L}(\mathcal{A}', \gamma'')$. Finally, we will compose γ' and γ'' to get γ.

We start by constructing γ''. According to Lemma 5 (w.l.o.g.) $B_k = \emptyset$. Consider the Rabin condition $\alpha'' = \{\langle G_k, \emptyset\rangle\}$. It is easy to see that α'' is equivalent to the parity condition $\gamma'' = \{\emptyset, G_k, Q, Q\}$. Intuitively, α'' and γ'' accept exactly all the words that could have been accepted thanks to the pair $\langle G_k, B_k\rangle$.

We now proceed to construct γ'. Let $\alpha' = \{\langle G_1, B_1 \cup G_k\rangle, \langle G_2, B_2 \cup G_k\rangle, \ldots, \langle G_{k-1}, B_{k-1} \cup G_k\rangle\}$. Intuitively, α' completes α'' by accepting all the words in $\mathcal{L}(\mathcal{A}, \alpha) \setminus \mathcal{L}(\mathcal{A}, \alpha'')$. It is easy to see that $\mathcal{L}(\mathcal{A}, \alpha') \cap \mathcal{L}(\mathcal{A}, \alpha'') = \emptyset$ and that $\mathcal{L}(\mathcal{A}, \alpha') = \mathcal{L}(\mathcal{A}, \alpha) \setminus \mathcal{L}(\mathcal{A}, \alpha'')$. Consider the Streett condition $\beta' = \{\langle L_1, U_1\rangle, \ldots, \langle L_l, U_l\rangle, \langle G_k, \emptyset\rangle\}$. We claim that β' is equivalent to α' on $\mathcal{A}$. That is, $\mathcal{L}(\mathcal{A}, \beta') = \mathcal{L}(\mathcal{A}, \alpha) \setminus \mathcal{L}(\mathcal{A}, \alpha'')$. To see that $\mathcal{L}(\mathcal{A}, \beta') \subseteq \mathcal{L}(\mathcal{A}, \alpha) \setminus \mathcal{L}(\mathcal{A}, \alpha'')$, consider a word $w \in \Sigma^\omega$, the single run r of $\mathcal{A}$ on w, and the set $inf(r)$. If $inf(r)$ satisfies β' than it clearly satisfies β and therefore $w \in \mathcal{L}(\mathcal{A}, \beta) = \mathcal{L}(\mathcal{A}, \alpha)$. Additionally, r satisfies the Streett pair $\langle G_k, \emptyset\rangle$ which implies that $inf(r) \cap G_k = \emptyset$, so r does not satisfy α' and therefore $w \notin \mathcal{L}(\mathcal{A}, \alpha')$. Showing the inclusion in the other way is similar.

Consider the pre-automaton $\mathcal{A}$ and the acceptance conditions α' and β'. The underlying graph of $\mathcal{A}$ is strongly connected and α' and β' are Rabin and Streett conditions such that $\mathcal{L}(\mathcal{A}, \alpha') = \mathcal{L}(\mathcal{A}, \beta')$ and the index of α' is $k-1$. In order, however, to apply the induction hypothesis, we also need $\mathcal{A}$ to be hopeless-free with respect to α' and β', which is clearly not the case, as the vertices in G_k are hopeless in $\mathcal{A}$ with respect to α'.

Let $\mathcal{C}$ denote the set of MSCCs of $G_{\mathcal{A}'}$, where $\mathcal{A}' = \mathcal{A}_{|Q'}$ and $Q' = Q \setminus \mathcal{H}_{\mathcal{A},\alpha'}$. According to Lemma 2, for every $C \in \mathcal{C}$ and for every $q \in C$ we have $\mathcal{L}(\mathcal{A}|_C^q, \alpha') = \mathcal{L}(\mathcal{A}|_C^q, \beta')$. Each $C \in \mathcal{C}$ is strongly connected and hopeless-free with respect to α'. Hence, the induction hypothesis implies that for each $C \in \mathcal{C}$ there is a parity condition γ^C of index at most $2k$, such that for every $q \in C$, we have $\mathcal{L}(\mathcal{A}|_C^q, \alpha') = \mathcal{L}(\mathcal{A}|_C^q, \gamma^C)$. According to Lemma 4, this implies the existence of a single condition $\gamma' = \{F'_1, \ldots, F'_{2k}\}$ such that $\mathcal{L}(\mathcal{A}, \alpha') = \mathcal{L}(\mathcal{A}, \gamma')$.

We define γ as the composition of γ' and γ''. Formally, $\gamma = \{\emptyset, G_k, F_3, \ldots, F_{2k+2}\}$ where for all $3 \leqslant i \leqslant m+2$, we set $F_i = F'_{i-2}$.

We show that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \gamma)$. Consider a word $w \in \Sigma^\omega$, the single run r of $\mathcal{A}$ on w, and the set $inf(r)$. If $w \in \mathcal{L}(\mathcal{A}, \alpha)$ then r satisfies α and there is an index $1 \leqslant i \leqslant k$ such that $inf(r) \cap G_i \neq \emptyset$ and $inf(r) \cap B_i = \emptyset$. If $inf(r) \cap G_k \neq \emptyset$ (i.e. $i = k$) then the minimal index for which $inf(r) \cap F_j \neq \emptyset$ is 2, which is even, and therefore r satisfies γ. Otherwise, r does not satisfy α'' and therefore since it is accepting and since $\mathcal{L}(\mathcal{A}, \alpha') = \mathcal{L}(\mathcal{A}, \alpha) \setminus \mathcal{L}(\mathcal{A}, \alpha'')$ it must satisfy α'. This, in turn, implies that r also satisfies γ'. Let j be the minimal index for which $inf(r) \cap F'_i \neq \emptyset$. Since r satisfies γ' the index j is even. Clearly, the minimal index for which $inf(r) \cap F_i \neq \emptyset$ equals $j+2$, and is therefore also even, and therefore r satisfies γ, thus $w \in \mathcal{L}(\mathcal{A}, \gamma)$ and we have $\mathcal{L}(\mathcal{A}, \alpha) \subseteq \mathcal{L}(\mathcal{A}, \gamma)$.

The other direction of the inclusion is similar. ◀

Lemmas 2 and 4 imply the generalization of Lemma 6 to any pre-automaton. Formally, we have the following.

▶ **Theorem 7.** *Let $\mathcal{A}$ be a deterministic pre-automaton with n states, α a Rabin condition of index k and β a Streett condition of index l such that $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\beta)$. Then, there exists a parity condition γ of index at most $\min\{2k+2, 2l+2, n+2\}$ such that $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\beta) = \mathcal{L}(\mathcal{A},\gamma)$.*

Proof. The proof follows a similar argumentation to the induction step in Lemma 6. Let $\mathcal{C}$ denote the set of MSCCs of $G_{\mathcal{A}'}$, where $\mathcal{A}' = \mathcal{A}_{|Q'}$ and $Q' = Q \setminus \mathcal{H}_{\mathcal{A},\alpha}$. According to Lemma 2, for every $C \in \mathcal{C}$ and for every $q \in C$ we have $\mathcal{L}(\mathcal{A}|_C^q, \alpha) = \mathcal{L}(\mathcal{A}|_C^q, \beta)$. Each $C \in \mathcal{C}$ is strongly connected and hopeless-free with respect to α. Hence, Lemma 6 implies that for each $C \in \mathcal{C}$ there is a parity condition γ^C of index at most $2k+2$, such that for every $q \in C$, we have $\mathcal{L}(\mathcal{A}|_C^q, \alpha') = \mathcal{L}(\mathcal{A}|_C^q, \gamma^C)$. According to Lemma 4, this implies the existence of a single condition $\gamma = \{F_1, \ldots, F_m\}$ of index at most $2k+2$ such that $\mathcal{L}(\mathcal{A},\alpha) = \mathcal{L}(\mathcal{A},\gamma)$.

By the above proof, the index of γ is at most $2k+2$. We now tighten it further. If $k > l$ we can switch the roles of α and β: Let $\tilde{\alpha}$ and $\tilde{\beta}$ be α and β when viewed as Streett and Rabin conditions, respectively. We still have $\mathcal{L}(\mathcal{A},\tilde{\alpha}) = \mathcal{L}(\mathcal{A},\tilde{\beta})$, recognizing the complementing language. By the above, there is a parity condition $\tilde{\gamma}$ of degree at most $2l+2$ such that $\mathcal{L}(\mathcal{A},\tilde{\gamma}) = \mathcal{L}(\mathcal{A},\tilde{\beta})$. In order to obtain a parity condition that would be equivalent to the original α, we dualize $\tilde{\gamma}$ (the way we have constructed $\tilde{\gamma}$ guarantees that a dualization would not involve an increase in the index). Finally, a parity condition on n states that has more than $n+2$ sets must contain equivalent subsequent sets and can therefore be simplified to one with at most $n+2$ sets. Hence the $\min\{2k+2, 2l+2, n+2\}$ bound. ◀

As discussed in Section 1, the considerations behind our proof are different than these used in [22] in the context of two-player games. In addition, our proof is constructive, and it generates the equivalent parity condition. It is not clear to us whether and how the proof in [22] can be adopted to the setting of automata. In particular, an attempt to generate a parity condition following the considerations in [22] involves an examination of subsets of the state space of the game, and is thus exponential. As we show below, our procedure requires only polynomial time.

PTIME-completeness. The proof above is constructive, allowing to generate the equivalent parity condition or return the answer that such a condition does not exist. We show below that our procedure is in PTIME and that the related question is indeed PTIME-complete.

▶ **Theorem 8.** *Consider a DRW or a DSW $\mathcal{A}$. The problem of deciding whether $\mathcal{A}$ has an equivalent DPW on the same structure is PTIME-complete.*

Proof. We prove the result for DRW. By the duality of the Rabin and Streett conditions, and the self-duality of the parity condition, the result for DSW follows.

We start with the upper bound. Assume that $\mathcal{A}$ has n states and its acceptance condition α is of index k. As discussed above, the given Streett condition does not play a role in the construction of the parity condition, and its essence is in guaranteeing the existence of a Rabin pair with an empty "bad" set. Accordingly, the procedure described above for generating γ works for every DRW and it always ends after up to $\min(n+1, k)$ iterations. Each iteration is clearly in PTIME, as it only marks the hopeless states, which can be done by exploring the loops in the automaton's graph. If it completes all iterations, then the DRW has an equivalent DPW on the same structure (the one generated by the procedure). Otherwise, the procedure gets stuck in an iteration in which no Rabin pair with an empty bad set exists, in which case the DRW does not have an equivalent DPW on the same structure.

It is left to prove PTIME-hardness. We do a reduction from DRW universality, which is dual to DSW emptiness, proved to be PTIME-complete in [6]. In the proof, we consider

languages over an alphabet $\Sigma_1 \times \Sigma_2$. For a word $w \in (\Sigma_1 \times \Sigma_2)^\omega$, let $w_1 \in \Sigma_1^\omega$ be the word obtained from w by projecting its letters on Σ_1, and similarly for w_2 and Σ_2. For words $x_1 \in \Sigma_1^\omega$ and $x_2 \in {\Sigma_2}^\omega$, let $x_1 \oplus x_2$ denote the word $w \in (\Sigma_1 \times \Sigma_2)^\omega$ with $w_1 = x_1$ and $w_2 = x_2$. Given a DRW $\mathcal{R}$ with alphabet Σ_1, we define another DRW $\mathcal{A}$ such that $\mathcal{R}$ is universal (that is, $L(\mathcal{R}) = \Sigma_1^\omega$) iff $\mathcal{A}$ has an equivalent DPW on the same structure. Let $\mathcal{R} = \langle \Sigma_1, Q, q_0, \delta, \alpha \rangle$, and let $\mathcal{R}' = \langle \Sigma_2, Q', q_0', \delta', \alpha' \rangle$ be a DRW such that there exists no DPW equivalent to $\mathcal{R}'$ on the same structure. We define $\mathcal{A} = \langle \Sigma_1 \times \Sigma_2, Q \times Q', \langle q_0, q_0' \rangle, \delta'', \alpha'' \rangle$, where

- $\delta''(\langle q, q' \rangle, \langle \sigma_1, \sigma_2 \rangle) = \langle \delta(q, \sigma_1), \delta'(q', \sigma_2) \rangle$, and
- $\alpha'' = \{\langle G \times Q', B \times Q' \rangle : \langle G, B \rangle \in \alpha\} \cup \{\langle Q \times G', Q \times B' \rangle : \langle G', B' \rangle \in \alpha'\}$.

It is easy to see that $\mathcal{L}(\mathcal{A}) = \{w \ : w_1 \in \mathcal{L}(\mathcal{R}) \text{ or } w_2 \in \mathcal{L}(\mathcal{R}')\}$. We prove that $\mathcal{R}$ is universal iff $\mathcal{A}$ has an equivalent DPW on the same structure. First, if $\mathcal{R}$ is universal, so is $\mathcal{A}$, and hence it clearly has an equivalent DPW on the same structure. Assume now that $\mathcal{R}$ is not universal, we show that there is no DPW equivalent to $\mathcal{A}$ on the same structure. Assume by way of contradiction that γ is a parity condition defined on top of $Q \times Q'$ such that the language of $\mathcal{A}$ with γ is equal to $\mathcal{L}(\mathcal{A})$. In the full version we prove that the projection of γ on Q' results in a parity condition γ' such that the language of $\mathcal{R}'$ with acceptance condition γ' is equivalent to $\mathcal{L}(\mathcal{R}')$. This, however, contradicts the assumption that no DPW equivalent to $\mathcal{R}'$ can be defined on the same structure. Essentially, the claim follows from the fact that if $w_1 \in \Sigma_1^\omega$ is a word rejected by $\mathcal{R}$ (since $\mathcal{R}$ is not universal, such a word exists), then the behavior of $\mathcal{A}$ on words whose projection on Σ_1 is w_1 depends only on its $\mathcal{R}'$ component. ◀

4 The Non-Deterministic Case

Can our results be generalized to the nondeterministic case? To show the converse we describe a nondeterministic pre-automaton $\mathcal{A}$ on top of which we define Rabin and Streett conditions, α and β, such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \beta)$, however there is no parity condition γ such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \gamma)$. It follows that our main result does not hold in the nondeterministic setting. Furthermore, by dualizing one gets a counterexample for the claim about universal automata (that is, alternating automata in which transitions are only conjunctively related). Indeed, the key role of the determinism in our proof is inevitable . We prove that the problem of deciding whether a given NRW or NSW has an equivalent NPW on the same structure is PSPACE-complete.

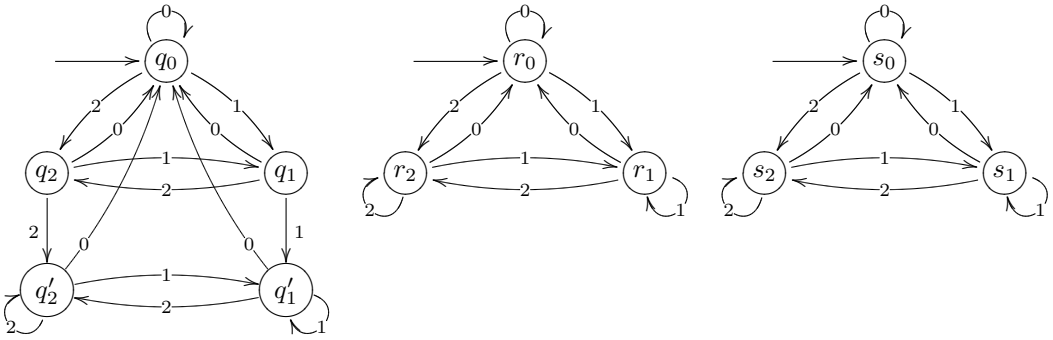


Figure 1 A nondeterministic pre-automaton $\mathcal{A}$ having equivalent Rabin and Streett conditions for L with no corresponding parity condition.

Consider the nondeterministic pre-automaton $\mathcal{A}$ over $\Sigma = \{0, 1, 2\}$ depicted in Figure 1. We use Q, R and S to denote the sets of states of the different components, thus $Q = \{q_0, q_1, q_1', q_2, q_2'\}$, $R = \{r_0, r_1, r_2\}$ and $S = \{s_0, s_1, s_2\}$. Note that the nondeterminism of $\mathcal{A}$ is

limited to the choice of initial state , thus it is a non-ambiguous (or a single-run) automaton. Consider the Rabin and Streett conditions

- $\alpha = \{\langle\{q_1\}, \{q_2, q_2'\}\rangle, \langle\{q_2\}, \{q_1, q_1'\}\rangle\}$ (Rabin), and
- $\beta = \{\langle Q \cup \{r_1, s_2\}, \emptyset\rangle, \langle R, \{r_0\}\rangle, \langle R, \{r_2\}\rangle, \langle S, \{s_0\}\rangle, \langle S, \{s_1\}\rangle\}$ (Streett).

We define $\mathcal{L} = \{w \in \Sigma^\omega : \mathit{inf}(w) = \{0,1\} \text{ or } \mathit{inf}(w) = \{0,2\}\}$ and show that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L} = \mathcal{L}(\mathcal{A}, \beta)$. It is easy to see that a word $w \in \mathcal{L}$ has an accepting run. In fact, the single run of $\mathcal{A}$ on w that is accepting with respect to α is the one that starts at q_0. On the other hand, a word $w \in \Sigma^\omega$ belongs to $\mathcal{L}(\mathcal{A}, \alpha)$ if there is a run r of $\mathcal{A}$ that satisfies α. Such a run must start at q_0, as otherwise r never visits any of α's "good" sets. Further, to satisfy the pair $\langle\{q_1\}, \{q_2, q_2'\}\rangle$ the run r must get trapped in the right part of Q, so w must contain only finitely many 2's and infinitely many 0's and 1's. Similarly, in order to satisfy the pair $\langle\{q_2\}, \{q_1, q_1'\}\rangle$, the run r must get trapped in the left part of Q, so it must consist of only finitely many 1's and infinitely many 0's and 2's.

Consider $\mathcal{L}(\mathcal{A}, \beta)$. It is easy to see that a word $w \in \mathcal{L}$ has an accepting run. In fact, if w has only finitely many 1's then the single run of $\mathcal{A}$ on w that is accepting with respect to β is the one that starts at r_0, and if w has only finitely many 2's then the single run of $\mathcal{A}$ on w that is accepting with respect to β is the one that starts at s_0. On the other hand, a word $w \in \Sigma^\omega$ belongs to $\mathcal{L}(\mathcal{A}, \beta)$ if there is a run r of $\mathcal{A}$ that satisfies β. Such a run must either start at r_0 or at s_0, as otherwise it will be trapped in Q and violate the pair $\langle Q \cup \{r_1, s_2\}, \emptyset\rangle$. If r starts at the state r_0, then in order to accept, w must consist of infinitely many 0's and 2's but only finitely many 1's. Otherwise, r must start at s_0, then in order to accept, w must consist of infinitely many 0's and 1's but only finitely many 2's.

We now show that there is no parity condition γ such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \gamma)$. Assume by contradiction that $\gamma = \{F_1, F_2, \ldots, F_m\}$ such that $\mathcal{L}(\mathcal{A}, \alpha) = \mathcal{L}(\mathcal{A}, \gamma)$. Referring to the minimal index i for which $q \in F_i$ by the *rank* or q, we note that all states with self loops cannot have an even rank, as otherwise words that consist of a single letter can have an accepting run. Hence if a run is accepting with respect to γ it must be contained in Q. Since $(01)^\omega \in \mathcal{L}(\mathcal{A}, \alpha)$ it must also be in $\mathcal{L}(\mathcal{A}, \gamma)$, therefore the run $(q_0, q_1)^\omega$ must be accepting with respect to γ. Since q_0 has a self loop it cannot be ranked evenly, therefore the rank of q_1 must be even. Similarly, q_2's rank must also be even. However, that would imply that the run $(q_1 q_2)^\omega$ on $(12)^\omega$ would be accepting with respect to γ, but $(12)^\omega \notin \mathcal{L}(\mathcal{A}, \alpha)$.

PSPACE-completeness. The counter example above suggests that the translation to an equivalent parity condition on the same structure is more complicated in the nondeterministic setting. Indeed, we show below that this problem is PSPACE-complete.

▶ **Theorem 9.** *Consider an NRW or an NSW $\mathcal{A}$. The problem of deciding whether $\mathcal{A}$ has an equivalent NPW on the same structure is PSPACE-complete.*

Proof. For the upper bound, one can go over all possible parity conditions for $\mathcal{A}$ and check the equivalence of the obtained NPW with $\mathcal{A}$. The lower bound is similar to the one described in the proof of Theorem 8, only that here the Rabin and Streett cases are not dual (dualizing an NRW, one gets a universal (rather than nondeterministic) Streett automaton), thus we have to consider both cases. In addition, for the lower bounds, while the reductions are still from the universality problem, now they are from NRW or NSW universality, which are PSPACE-complete. ◀

References

1 J.R. Büchi. On a decision method in restricted second order arithmetic. In *Proc. Int. Congress on Logic, Method, and Philosophy of Science. 1960*, pages 1–12, 1962.

2 Y. Choueka. Theories of automata on ω-tapes: A simplified approach. *Journal of Computer and Systems Science*, 8:117–141, 1974.

3 C. Eisner and D. Fisman. *A Practical Introduction to PSL*. Springer, 2006.

4 A.E. Emerson and A.P. Sistla. Deciding full branching time logics. *Information and Control*, 61(3):175–201, 1984.

5 E.A. Emerson and C. Jutla. Tree automata, μ-calculus and determinacy. In *Proc. 32nd IEEE Symp. on Foundations of Computer Science*, pages 368–377, 1991.

6 E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. 18th Hawaii Int. Conf. on System Sciences*, 1985.

7 Y. Gurevich and L. Harrington. Trees, automata, and games. In *Proc. 14th ACM Symp. on Theory of Computing*, pages 60–65. ACM Press, 1982.

8 M. Henzinger and J.A. Telle. Faster algorithms for the nonemptiness of Streett automata and for communication protocol pruning. In *Proc. 5th Scandinavian Workshop on Algorithm Theory*, volume 1097 of *LNCS*, pages 10–20. Springer, 1996.

9 M. Jurdzinski, M. Paterson, and U. Zwick. A deterministic subexponential algorithm for solving parity games. *SIAM Journal on Computing*, 38(4):1519–1532, 2008.

10 V. King, O. Kupferman, and M.Y. Vardi. On the complexity of parity word automata. In *Proc. 4th Int. Conf. on Foundations of Software Science and Computation Structures*, volume 2030 of *LNCS*, pages 276–286. Springer, 2001.

11 S.C. Krishnan, A. Puri, and R.K. Brayton. Deterministic ω-automata vis-a-vis deterministic Büchi automata. In *Algorithms and Computations*, volume 834 of *LNCS*, pages 378–386. Springer, 1994.

12 O. Kupferman, G. Morgenstern, and A. Murano. Typeness for ω-regular automata. *International Journal on the Foundations of Computer Science*, 17(4):869–884, 2006.

13 O. Kupferman and M.Y. Vardi. Safraless decision procedures. In *Proc. 46th IEEE Symp. on Foundations of Computer Science*, pages 531–540, 2005.

14 O. Kupferman, M.Y. Vardi, and P. Wolper. An automata-theoretic approach to branching-time model checking. *Journal of the ACM*, 47(2):312–360, 2000.

15 R.P. Kurshan. *Computer Aided Verification of Coordinating Processes*. Princeton Univ. Press, 1994.

16 C. Löding. Optimal bounds for the transformation of omega-automata. In *Proc. 19th FSTTCS Conference*, volume 1738 of *LNCS*, pages 97–109, 1999.

17 A.W. Mostowski. Regular expressions for infinite trees and a standard form of automata. In *Computation Theory*, volume 208 of *LNCS*, pages 157–168. Springer, 1984.

18 N. Piterman. From nondeterministic Büchi and Streett automata to deterministic parity automata. In *Proc. 21st LICS conference*, pages 255–264. IEEE press, 2006.

19 S. Safra. On the complexity of ω-automata. In *Proc. 29th FOCS*, pages 319–327, 1988.

20 S. Safra and M.Y. Vardi. On ω-automata and temporal logic. In *Proc. 21st ACM Symp. on Theory of Computing*, pages 127–137, 1989.

21 M.Y. Vardi and P. Wolper. Reasoning about infinite computations. *Information and Computation*, 115(1):1–37, 1994.

22 W. Zielonka. Infinite games on finitely colored graphs with applications to automata on infinite trees. *Theoretical Computer Science*, 200(1-2):135–183, 1998.

Finding Sparser Directed Spanners*

Piotr Berman[1], Sofya Raskhodnikova[1], and Ge Ruan[1]

1 **Pennsylvania State University, USA**
{berman,sofya,gur112}@cse.psu.edu

Abstract

A spanner of a graph is a sparse subgraph that approximately preserves distances in the original graph. More precisely, a subgraph $H = (V, E_H)$ is a *k-spanner* of a graph $G = (V, E)$ if for every pair of vertices $u, v \in V$, the shortest path distance $dist_H(u, v)$ from u to v in H is at most $k \cdot dist_G(u, v)$. We focus on spanners of *directed graphs* and a related notion of *transitive-closure spanners*. The latter captures the idea that a spanner should have a small diameter but preserve the connectivity of the original graph. We study the computational problem of finding the sparsest k-spanner (resp., k-TC-spanner) of a given *directed* graph, which we refer to as Directed k-Spanner (resp., k-TC-Spanner). We improve all known approximation algorithms for Directed k-Spanner and k-TC-Spanner for $k \geq 3$. (For $k = 2$, the current ratios are tight, assuming P≠NP.) Along the way, we prove several structural results about the size of the sparsest spanners of directed graphs.

Keywords and phrases Approximation algorithms, directed graphs, spanners

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.424

1 Introduction

A spanner of a graph is a sparse subgraph that approximately preserves distances in the original graph.

▶ **Definition 1.1** (k-spanner, [3, 24])**.** A subgraph $H = (V, E_H)$ is a **k-spanner** of a graph $G = (V, E)$ if for all vertices $u, v \in V$, the shortest path distance $dist_H(u, v)$ from u to v in H is at most $k \cdot dist_G(u, v)$. The parameter k is called the **stretch**.

Graph spanners have numerous applications, ranging from efficient routing [13, 14, 26, 31] and simulating synchronized protocols in unsynchronized networks [25] to parallel and distributed algorithms for approximating shortest paths [11, 12, 15] and algorithms for distance oracles [5, 32].

We focus on spanners of *directed graphs* and a related notion of *transitive-closure spanners* studied in [7, 28, 29, 30, 19, 8, 6]. The latter captures the idea that a spanner should have a small diameter but preserve the connectivity of the original graph. The diameter here means the largest distance between a pair (u, v) of nodes in a directed graph such that v is reachable from u. Recall that the *transitive closure* of a graph $G = (V, E)$, denoted $TC(G)$, is a graph (V, E_{TC}) where $(u, v) \in E_{TC}$ if and only if G has a directed path from u to v.

▶ **Definition 1.2** (k-TC-spanner, [7])**.** A directed graph $H = (V, E_H)$ is a **k-transitive-closure-spanner** (**k-TC-spanner**) of a directed graph $G = (V, E)$ if

1. E_H is a subset of the edges in the transitive closure of G and

* S.R. was supported by National Science Foundation (NSF/CCF award 0729171 and NSF/CCF CAREER award 0845701).

2. for all vertices $u, v \in V$, if $dist_G(u, v) < \infty$, then $dist_H(u, v) \leq k$.

The edges from the transitive closure of G that are added to G to obtain a TC-spanner are called **shortcuts**.

Notice that a k-TC-spanner of G is a directed k-spanner of the transitive closure of G. Nevertheless, TC-spanners are interesting in their own right due to the multiple TC-spanner-specific applications, such as managing keys in access control hierarchies, data structures for computing partial products in a semigroup, property testing and property reconstruction (see the survey in [27] and references therein).

In this paper, we study the computational problem of finding the sparsest k-TC-spanner (resp., k-spanner) of a given *directed* graph, which we refer to as k-TC-Spanner (resp., Directed k-Spanner).

1.1 Previous Work

To put the following results in proper context, observe that if a graph G has n vertices, every k-spanner of G has $O(n^2)$ edges.

1.1.1 Computational Results

All algorithms for Directed k-Spanner immediately yield algorithms for k-TC-Spanner with the same approximation ratio because k-TC-Spanner on input graph G is equivalent to Directed k-Spanner on input TC(G). Elkin and Peleg [17] gave an $O(\log n)$-approximation algorithm for Directed 2-Spanner. Kortsarz [22] showed that the $O(\log n)$ approximation ratio for Directed 2-Spanner cannot be improved unless P=NP, and [7] extended this result to 2-TC-spanner. Thus, for $k = 2$ the ratio for both problems has been completely resolved.

For $k = 3$, the first non-trivial approximation ratio for Directed k-Spanner was proved by Elkin and Peleg [16] and slightly improved (by a factor polylogarithmic in n) by Bhattacharyya *et al.* [7]. In general, for $k \geq 3$, [7] gives an approximation ratio $O((n \log n)^{1-1/k})$. For all $\delta, \epsilon \in (0, 1)$ and $3 \leq k \leq n^{1-\delta}$, it is impossible to approximate Directed k-Spanner within a factor of $2^{\log^{1-\epsilon} n}$ in polynomial time, assuming NP$\not\subseteq$DTIME($n^{\text{polylog}\, n}$) [16, 18]. ($DTIME(f(n))$ denotes the class of languages decidable deterministically in time $f(n)$.) Thus, according to Arora and Lund's classification [20] of NP-hard problems, Directed k-Spanner is in class III, for $k \in [3, n^{1-\delta}]$. [7] extended this result to k-TC-Spanner for constant k. [18] also showed that proving that Directed k-Spanner is in class IV, that is, inapproximable within n^δ for some $\delta \in (0, 1)$, would resolve a long standing open question in complexity theory: namely, cause classes III and IV to collapse into a single class. Since k-TC-Spanner is a special case of Directed k-Spanner, this statement also applies to the former.

For the special case of k-TC-Spanner, [7] presented an $O\left(\frac{n \log n}{k^2 + k \log n}\right)$-approximation algorithm. Observe that for large k, this is better than the inapproximability of Directed k-Spanner, demonstrating that k-TC-Spanner is a strictly easier problem for this range of parameters.

1.1.2 Structural Results

Unlike in the undirected setting, where for every $k \geq 1$, all graphs on n vertices have $(2k-1)$-spanners with $O(n^{1+1/k})$ edges [2, 23, 32], sparsest TC-spanners (and hence, sparsest directed spanners) can have $\Omega(n^2)$ edges. An example of a graph with a sparsest 2-TC-spanner of

Problem	Stretch k	Hardness	Known Ratio	Our Ratio
Directed spanner and TC-spanner	2	$\Omega(\log n)$ [22, 7]	$O(\log n)$ [17]	–
	3	$\Omega(2^{\log^{1-\epsilon} n})$ [16, 18, 7]	$O((n \log n)^{2/3})$ [16, 7]	$O(\sqrt{n} \log n)$
Directed Spanner	> 3	as above, for $k \leq n^{1-\delta}$ [16, 18]	$O((n \log n)^{1-1/k})$ [7]	$O(k \cdot n^{1-1/\lceil k/2 \rceil} \log n)$
TC-spanner	> 3	as above, for constant k [7]	as above	$O(n^{1-1/\lceil k/2 \rceil} \log n)$
	$\Omega\left(\frac{\log n}{\log\log n}\right)$	$(1+\delta)$, for $k \leq n^{1-\delta}$ [7]	$O\left(\frac{n \log n}{k^2 + k \log n}\right)$ [7]	$O\left(\frac{n}{k^2}\right)$

Table 1 Summary of Results on Approximability of k-TC-Spanner and Directed k-Spanner

size $\Omega(n^2)$ is the complete bipartite graph $K_{\frac{n}{2},\frac{n}{2}}$ with $n/2$ vertices in each part and all edges directed from the first part to the second. Therefore, to the best of our knowledge, there are no combinatorial results about the sizes of sparsest spanners for general directed graphs.

There are many combinatorial results on the sizes of the sparsest k-TC-spanners but, for the reason mentioned above, almost all of them apply exclusively to special families of graphs: directed lines and trees [10, 30], planar graphs [28], H-minor-free graphs [7], multi-dimensional grids [6] and low-dimensional posets [9]. There are only two results that can be viewed as structural results for general graphs. The first one is the TC-spanner-specific algorithm from [7], which is obtained by showing that every graph can be turned into a k-TC-spanner by adding $O\left(\frac{n^2 \log n}{k^2 + k \log n}\right)$ shortcut edges. The second one is a construction of Hesse [19] giving a family of graphs with large TC-spanners and disproving Thorup's conjecture [28] that all directed graphs G on n nodes have TC-spanners with stretch polylogarithmic in n and size at most $2|G|$. (We use $|G|$ to denoted the number of edges in G.) For all small $\epsilon > 0$, Hesse constructed a family of graphs with $n^{1+\epsilon}$ edges for which all n^{ϵ}-TC-spanners require $\Omega(n^{2-\epsilon})$ edges.

1.2 Our Results and Techniques

1.2.1 Computational Results

Table 1 summarizes the best known results on approximability of k-TC-Spanner and Directed k-Spanner, alongside with our results on these problems. We improve all known approximation algorithms for Directed k-Spanner and k-TC-Spanner for $k \geq 3$. (As mentioned before, for $k = 2$, the current ratios are tight, assuming P$\neq$NP.)

First, we obtain $O(k \cdot n^{1-1/\lceil k/2 \rceil} \log n)$-approximation for Directed k-Spanner. In particular, for both Directed k-Spanner and k-TC-Spanner, the best previously known ratios were $O((n \log n)^{2/3})$ for $k = 3$ and $O((n \log n)^{3/4})$ for $k = 4$. Our algorithm improves both of these ratios to $O(\sqrt{n} \log n)$. Second, we give a slightly better $O(n^{1-1/\lceil k/2 \rceil} \log n)$-approximation for k-TC-Spanner. Finally, we also present a $O(n/k^2)$-approximation algorithm for k-TC-Spanner, which outperforms our first algorithm for $k = \Omega(\log n / \log\log n)$.

1.2.2 Our Techniques and Structural Results

The approach we take is very different from previous work. The best previously known algorithm for small $k > 2$ from [7] (that works for k-TC-Spanner and Directed k-Spanner) is based on solving a linear program and combining the solution with paths obtained by random sampling. Our approximation algorithm for k-TC-Spanner for small k is based on the following idea: since known approximation algorithms do much better for stretch $k = 2$, we use a 2-TC-spanner produced by the approximation algorithm for 2-TC-spanner to approximate the sparsest k-TC-spanner. To show that it provides a good approximation, we prove that a k-TC-spanner of G cannot be much sparser than a 2-TC-spanner of G. More precisely, we show how to transform a k-TC-spanner into a 2-TC-spanner while increasing the number of edges by a factor of $O(n^{1-1/\lceil k/2 \rceil})$. These results are presented in Section 2.

In Section 3, we present our algorithm for Directed k-Spanner. First, we note that the idea used to approximate k-TC-Spanner cannot be applied directly: as shown in Lemma 3.2, there are directed graphs where the ratio between the sizes of the sparsest $(k-1)$-spanner and k-spanner is $\Omega(n)$. Instead, we build on the ideas of [21, 17] which use star graphs to give an approximation algorithm for Directed 2-spanner. We generalize the notion of a star to k-stars. Recall that stars contain a central node s and nodes of distance 1 from s. Intuitively, k-stars also have a central node s and nodes of distance at most k from s. We show that a k-spanner of a directed graph G can be approximated well by a collection of k-stars such that each edge (u, v) of G is *covered* by some k-star in the collection—namely, that k-star contains a path of length at most k from u to v. Then we use a greedy algorithm to find such a collection of k-stars with nearly minimum cost, where the cost is the total number of edges in all k-stars in the collection.

Our algorithm for Directed k-Spanner also works for stretch $k = 2$. Even though an algorithm with the same (optimal) approximation ratio has already been presented by Elkin and Peleg [17], our algorithm gives a unified treatment for all small k, including 2, and is slightly simpler for the case of $k = 2$ than the algorithm in [17]. (Elkin and Peleg use a slightly different greedy method to find stars. They categorize edges already covered by selected stars into different types and calculate two types of density, φ-density and ρ-density, to evaluate the benefit of adding a new star to the partial 2-spanner.)

In Section 4, we show that our transformation from a k-TC-spannner to a 2-TC-spanner cannot be improved significantly. We show that this transformation is tight for $k = 3$ and $k = 4$ by giving a simple example. For $k \geq 5$, we prove that our transformation is nearly tight by extending Hesse's construction [19], which was originally devised to disprove Thorup's conjecture, as described above. For every sufficiently small positive ϵ, we exhibit graphs with $2k$-TC-spanners of size $n^{1+1/k}$ and no 2-TC-spanners of size less than $n^{2-\epsilon}$.

Our approximation algorithm for k-TC-Spanner for large k, presented in Section 5, is also based on a structural result. Namely, we show how to (efficiently) obtain a k-TC-spanner of any graph by adding $O(n^2/k^2)$ shortcut edges. This is based on a simple greedy procedure. The algorithm for large stretch in [7] was based on random sampling.

1.3 Preliminaries

For a directed graph G, we denote the number of edges in G by $|G|$ and the size of the sparsest k-TC-spanner of G by $S_k(G)$. (The size refers to the number of edges.) We say two nodes in G are *comparable* if one of these nodes is reachable from the other.

A digraph G is *weakly connected* if replacing each directed edge in G with an undirected edge results in a connected undirected graph. A digraph is *strongly connected* if each

vertex in the graph is reachable from every other vertex via a directed path. The graph of strongly connected components of a digraph G is the digraph obtained by contracting each strongly connected component into one vertex, while maintaining all the edges between these components.

A *transitive reduction* of G is a digraph G' with the fewest edges for which $TC(G') = TC(G)$. As shown by Aho *et al.* [1], a transitive reduction of a given graph can be computed efficiently via a greedy algorithm. The algorithm contracts each strongly connected component C to a vertex $v(C)$ to get a supergraph H, obtains a supergraph H' by greedily removing edges in H that do not change its transitive closure, and finally uncontracts $v(C)$ to an arbitrary directed cycle on vertices in C, choosing a representative vertex of C to be incident to the edges incident to $v(C)$ in H'. Directed acyclic graphs have a unique transitive reduction. We say G is *transitively reduced* if G is equal to its own transitive reduction.

2 Approximation Algorithm for k-TC-Spanner for small k

This section presents an approximation algorithm for k-TC-Spanner that provides the best known ratio for all stretches $k = O(\log n / \log\log n)$.

▶ **Theorem 2.1.** *k-TC-Spanner can be approximated with ratio $O(n^{1-1/\lceil k/2\rceil}\log n)$ in polynomial time.*

The proof of this theorem appears at the end of this section. It relies on the following lemma and corollary that relate $S_k(G)$ and $S_2(G)$.

▶ **Lemma 2.2.** *Let G be a directed graph of diameter at most k. Then one can efficiently construct a 2-TC-spanner of G while increasing the number of edges by a factor of $O(n^{1-1/\lceil k/2\rceil})$.*

Proof. Let G be a directed graph on n nodes of diameter at most k. Set $d = n^{1/\lceil k/2\rceil}$. We call a vertex in G *high-degree* if its degree is at least d, and *low-degree* otherwise. (The degree of a node counts both incoming and outgoing edges.)

Transformation from k-TC-spanners to 2-TC-spanners
Input: directed graph $G = (V, E)$ of diameter k
1. Form graph H from G by adding shortcut edges as follows:
   a. Connect every *high-degree* vertex u to all vertices comparable to u.
   b. Consider the induced subgraph G' of G containing only *low-degree* vertices. Connect every vertex u in G' to all vertices that are reachable from u by a path of length at most $\lceil k/2\rceil$ in G'.
2. Output H.

To see that the resulting graph H is a 2-TC-spanner, consider a pair of vertices (u, v) with v reachable from u in G. Let P be a shortest path from u to v in G. If P contains a *high-degree* vertex w then in step 1(a) of the transformation, u was connected to v by a path of length at most 2 via w. Otherwise, all nodes in P are *low-degree*. Since G has diameter at most k, path P contains a node w such that $dist_G(u, w) \le \lceil k/2\rceil$ and $dist_G(w, v) \le \lceil k/2\rceil$. Then in step 1(b) of the transformation, u was connected to v by a path of length at most 2 via w. Thus, every comparable pair of nodes in H is connected by a path of length 2 or 1.

Since the number of edges in a graph is 1/2 of the sum of its vertex degrees, to prove the stated bound on $|H|/|G|$, it is enough to show that the degree of each vertex increases by a factor of $O(n^{1-1/\lceil k/2\rceil})$ when H is constructed from G. This holds for each *high-degree* node

because its degree increases from at least $d = n^{1/\lceil k/2 \rceil}$ to at most $2(n-1)$ (counting both incoming and outgoing edges). Each *low-degree* vertex v of degree $deg(v)$ connects to at most $deg(v) \cdot d$ vertices at distance 2, $deg(v) \cdot d^2$ vertices at distance 3, ..., $deg(v) \cdot d^{\lceil k/2 \rceil - 1}$ vertices at distance $\lceil k/2 \rceil$. So, the degree of v in H is at most $deg(v) \cdot (d^{\lceil k/2 \rceil} - 1)/(d-1)$, thus increasing by a factor of $O(d^{\lceil k/2 \rceil - 1}) = O(n^{(\lceil k/2 \rceil - 1)/\lceil k/2 \rceil}) = O(n^{1-1/\lceil k/2 \rceil})$, as desired. ◀

We obtain the following corollary by letting G in Lemma 2.2 be the sparsest k-TC-spanner of a given graph.

▶ **Corollary 2.3.** $\frac{S_2(G)}{S_k(G)} = O(n^{1-1/\lceil k/2 \rceil})$ *for any digraph* G. *In particular,* $\frac{S_2(G)}{S_3(G)} = O(\sqrt{n})$.

Proof of Theorem 2.1. Elkin and Peleg [17] gave an $O(\log n)$-approximation algorithm for Directed 2-spanner. Running this algorithm on the transitive closure of an input graph G will yield an $O(\log n)$-approximation to the sparsest 2-TC-spanner of G, which, by Corollary 2.3, is a $O(n^{1-1/\lceil k/2 \rceil} \log n)$-approximation to the sparsest k-TC-spanner of G. ◀

3 Approximation Algorithm for Directed k-Spanner

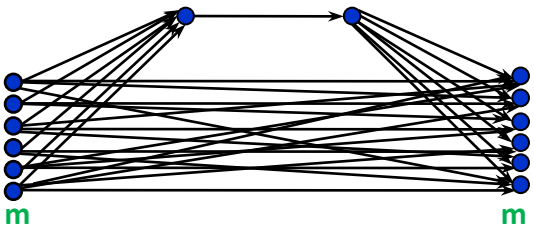
In this section, we present our approximation algorithm for Directed k-Spanner.

▶ **Theorem 3.1.** Directed k-Spanner *can be approximated with ratio* $O(k \cdot n^{1-1/\lceil k/2 \rceil} \log n)$ *in polynomial time.*

The following lemma demonstrates that the idea of approximating a k-spanner with a 2-spanner does not work for Directed k-Spanner.

▶ **Lemma 3.2.** *There is an infinite family of directed graphs* G *on* n *nodes for which every 2-spanner has* $\Omega(n^2)$ *edges while there is a* k*-spanner with* $O(n)$ *edges.*

Proof. Let m be a parameter such that $n = 2m + 2$. Our graph G has four layers of nodes, L_1, L_2, L_3 and L_4, of sizes $m, 1, 1$, and m, respectively. All edges are directed from smaller to larger layers. The following pairs of layers form complete bipartite graphs: (L_1, L_2), (L_2, L_3), (L_3, L_4) and (L_1, L_4). This completes the description of G.



G is the sparsest 2-spanner itself. It has $m^2 + 2m + 1 = \Omega(n^2)$ edges. To form a 3-spanner of G, we can delete all edges between layers L_1 and L_4. The resulting 3-spanner has $n - 1 = O(n)$ edges. By definition, a 3-spanner is also a k-spanner for all $k \geq 3$. ◀

Thus, approximating the sparsest k-spanner with a sparse 2-spanner will give an $\Omega(n)$ ratio in the worst case. Instead, we build on the ideas of [21, 17] which use star graphs to give an approximation algorithm for Directed 2-spanner.

▶ **Definition 3.3** (ℓ-Star, full ℓ-Star)**.** For a digraph G, a *1-star* is a complete 3-layered subgraph of G where all edges are directed from smaller to larger layers and the 2nd layer consists of a single node v. (Layers 1 and 3 can be empty.) More generally, an *ℓ-star* S_v is a $(2\ell + 1)$-layered subgraph of G where all edges start at some layer and end at the next

layer, and the middle layer consists of a single node v, called the *center*. Moreover, nodes in layers 1 to $\ell+1$ and edges between them form a shortest path tree (with respect to G) with root v (and edges directed towards the root). Similarly, nodes in layers $\ell+1$ to $2\ell+1$ and edges between them form a shortest path tree with root v (and edges directed away from the root). Some layers, can be empty, as long as the shortest path trees are valid. An ℓ-star that includes all vertices in G at distance at most ℓ from (to) the center is called *full*.

A full ℓ-star with center v can be found by performing two breadth-first searches on G starting from v (one on outgoing edges, another on incoming edges) and terminating both when all nodes at distance at most ℓ from (to) v have been found. Note that ℓ'-star is also an ℓ-star if $\ell' \leq \ell$. When ℓ is not specified or clear from the context, we omit it and call the corresponding graph a *star*.

In the context of finding directed k-spanners, we say that a star S *covers* an edge (u, v) of G if S contains a path of length at most k from u to v. A *covering* set C of stars is a set of stars such that each edge in G is covered by at least one of the stars in C. The *cost* of a set C of stars, denoted $cost(C)$, is the sum of the sizes of stars in the set. Observe that by taking all edges in the stars of a covering set C of stars, we obtain a directed k-spanner of size at most $cost(C)$.

Given a directed 2-spanner H of G, it is easy to construct a set C of 1-stars that cover every edge in G with $cost(C) \leq 2|H|$. E.g., we can include the full 1-star S_v with respect to H for each node in H. Since H is a 2-spanner of G, each edge (u, v) in G has a path of length at most 2 in H. This path is included in at least one of the stars in our set. Therefore, each edge is covered by one of the stars in the set. Since each edge is included in at most 2 stars, $cost(C) \leq 2|H|$.

The main idea of our algorithm for Directed k-Spanner is to cover all edges of G with stars. The next lemma shows that a covering set of k-stars with relatively low cost always exists.

▶ **Lemma 3.4.** *Let G be a directed graph on n nodes, and H be a k-spanner of G. Then there is a set C of k-stars that cover all edges in G with $cost(C) = O(n^{1-1/\lceil k/2 \rceil}|H|)$.*

Proof. We build on ideas in the proof of Lemma 2.2. As before, set $d = n^{1/\lceil k/2 \rceil}$. We call a vertex *high-degree* if its degree in H is at least d, and *low-degree* otherwise.

Transformation from a k-spanner to a covering set of stars

Input: a k-spanner H of G

1. Form a collection C of stars as follows:
   a. For every *high-degree* vertex u, add a full (with respect to H) k-star with center u.
   b. Consider the induced subgraph H' of H containing only *low-degree* vertices. For every vertex u in H', add a $\lceil k/2 \rceil$-star with center u containing all vertices v with $dist_{H'}(u, v)$ or $dist_{H'}(v, u)$ at most $\lceil k/2 \rceil$.
2. Output C.

First, we show that every edge (u, v) in G is covered by a star in C. Let P be a shortest path (of length at most k) from u to v in H. If P contains a *high-degree* vertex w then in step 1(a) of the transformation, a full k-star S_w with center w was added to C. By definition, S_w must contain both u and v as well as shortest paths from u to w and from w to v. Since w is on a shortest path u to v, it also contains a shortest path from u to v, and thus covers (u, v).

Otherwise, all nodes on path P are *low-degree*. Let w be the "middle" node on path P: that is, a node such that $dist_P(u,w) \leq \lceil k/2 \rceil$ and $dist_P(w,v) \leq \lceil k/2 \rceil$. Then in step 1(b) of the transformation, set C got a $\lceil k/2 \rceil$-star S_w with center w containing all nodes on path P. Since w is on a shortest path from u to v, this $\lceil k/2 \rceil$-star contains a shortest path from u to v via w. Thus, every edge in G is covered by a star in C.

Now we analyze $cost(C)$. Let $deg(u)$ denote the degree of node u in H. We add one star S_u to H for each node u. If u is a *high-degree* node, S_u has at most $2n = O(n^{1-1/\lceil k/2 \rceil}) \cdot deg(u)$ edges. Each *low-degree* vertex u connects to at most $deg(u)$ vertices at distance 1, $deg(u) \cdot d$ vertices at distance 2, $deg(u) \cdot d^2$ vertices at distance 3, ..., $deg(u) \cdot d^{\lceil k/2 \rceil - 1}$ vertices at distance $\lceil k/2 \rceil$. So, the number of edges in S_u is $O(d^{\lceil k/2 \rceil - 1}) \cdot deg(v) = O(n^{1-1/\lceil k/2 \rceil}) \cdot deg(u)$. Therefore, $cost(C) = \sum_{v \text{ in } H} |S_v| = O(n^{1-1/\lceil k/2 \rceil}) \cdot \sum_{v \text{ in } H} deg(u) = O(n^{1-1/\lceil k/2 \rceil}|H|)$. ◀

Next we prove the main theorem of this section.

Proof of Theorem 3.1. Our algorithm for Directed k-Spanner works by greedily constructing a covering set C of stars for the input graph G. As we mentioned, such a set easily yields a directed k-spanner of G of size $cost(C)$. Observe that the problem of finding a covering set of stars of minimum cost is a special case of Weighted Set Cover, where edges of G correspond to elements to be covered, and each star with $|S|$ edges corresponds to a covering set of weight (or cost) $|S|$. There are too many possible stars to write down the instance of Weighted Set Cover. However, recall that a simple greedy algorithm that chooses the set with the smallest relative cost (*i.e.,* the ratio of the cost of the set over the number of new elements it covers) gives approximation ratio $O(\log n)$ to Weighted Set Cover [33]. Therefore, if we had a subroutine that finds a star with the smallest relative cost in polynomial time, we would have a $O(\log n)$-approximation to our star covering problem. Instead, we will give a subroutine that finds a k-approximation to a star with the smallest relative cost, that is, it finds a star with relative cost at most k times the optimum. Then, the standard analysis of the greedy algorithm for Weighted Set Cover yields approximation ratio $O(k \log n)$.

It is not hard to see that a 1-star S_v with center v with smallest relative cost can be found in polynomial time using flow techniques (e.g., Project Selection). By finding such a star S_v for each v in G and selecting the best star, we can find the star with smallest relative cost. Incidentally, together with Lemma 3.4 and the observations above about Weighted Set Cover, this procedure gives a $O(\log n)$ approximation algorithm for Directed 2-Spanner.

Now let *opt* be the smallest relative cost of a k-star with center v. (Recall that an ℓ-star is also a k-star for $\ell \leq k$.) We will show how to find an k-star S_v with center v and relative cost at most $k \cdot opt$. Let G' be a graph obtained from G by adding edges from v (resp., to v) to all nodes reachable from v (resp., from all nodes from which v is reachable) at distance at most k. We run the subroutine above for finding a 1-star with center v of smallest relative cost on G'. It is guaranteed to output a 1-star of relative cost at most *opt*. This 1-star corresponds to a k-star in G of relative cost at most $k \cdot opt$, since the cost of adding each node is at most k instead of 1. As before, we run this subroutine for all nodes v and select the best star to obtain a k-approximation to a k-star with smallest relative cost.

We use the above subroutine to select k-stars for our collection C until all edges of G are covered. Let OPT be the smallest cost of a covering set of k-stars. As we explained, our algorithm will produce a covering set C of stars of cost $O(OPT \cdot k \log n)$. Let H be a sparsest k-spanner of G. By Lemma 3.4, there is a covering set C^* of k-stars with $cost(C^*) = O(n^{1-1/\lceil k/2 \rceil}|H|)$. Therefore, $cost(C) = O(cost(C^*) \cdot k \log n) = O(k \cdot n^{1-1/\lceil k/2 \rceil}(\log n)|H|)$, as required. ◀

4 Tightness of k-TC-spanner to 2-TC-spanner Transformation

In this section, we show that our transformation from a k-TC-spanner to a 2-TC-spanner (specifically, Corollary 2.3) is nearly tight. The main result of this section, Theorem 4.1, is proved by a construction that adapts ideas of Hesse [19]. At the end of the section, we present a simple direct construction that gives a stronger result for $k = 3$ and $k = 4$: in Lemma 4.5, we demonstrate that the transformation from a 3-TC-spanner to a 2-TC-spanner is asymptotically tight. Since our transformation to 2-TC-spanners incurs the same increase in the number of edges whether we start from 3- or 4-TC-spanners, and since $S_4(G) \leq S_3(G)$, Lemma 4.5 also implies the tightness of the transformation from 4-TC-spanners.

▶ **Theorem 4.1.** *For all $k > 2$ and $\epsilon > 0$, there exists an infinite family of graphs G on n nodes such that $S_2(G)/S_{2k}(G) \geq n^{1-1/k-\epsilon}$.*

Proof. The crux of our construction is the same as Hesse's [19]. The main difference is that he was trying to get an extreme example with a large number of layers, while our goal is to keep the number of layers small. Consequently, Hesse did not analyze our variant of his construction.

▶ **Definition 4.2** (Sets $C_{r,d}$ and $F_{r,d}$, [19])**.**

- $C_{r,d}$ is the set of vectors in $\mathbb{Z}^d$ of Euclidean length at most r.
- $F_{r,d}$ is the set of extremal points, "corners", of the convex hull of $C_{r,d}$.

Now we describe our graph, $G = G(d, r, k)$. It has some similarity to the Butterfly Network: in particular, its nodes are of the form $(\ell, x_1, \ldots, x_k)$ and its edges are of the form $((\ell - 1, x_1, \ldots, x_k), (\ell, x'_1, \ldots, x'_k))$ where ℓ is the node level, an integer in the range from 0 to $2k$. The remaining k coordinates are called *data coordinates*. Like in the Butterfly network, only one data coordinate changes when an edge is traversed, as indicated by the ℓ-coordinate: namely, $x_i = x'_i$ if $i \neq \ell$ and $i \neq k + \ell$. However, the range of data coordinates and the allowed changes are different. In particular, before the first allowed change, each data coordinate has range $C_{r,d}$, after the first change the allowed range is $C_{2r,d}$ and after the second change, the range is $C_{3r,d}$. The allowed change is $x'_i = x_i + z_i$ where $z_i \in F_{r,d}$.

Observe that G is its own $2k$-TC-spanner since every directed path has at most $2k$ edges. That is, $S_{2k}(G) = |G|$. One can also see that the out-degree of every node, except for nodes in layer $2k$, is $|F_{r,d}|$.

In the next lemma, we analyze paths between the first and the last layer of G.

▶ **Lemma 4.3.** *For all $x = (x_1, \ldots, x_k) \in (C_{r,d})^k$ and all $z = (z_1, \ldots, z_k) \in (F_{r,d})^k$, there exists exactly one path from $(0, x)$ to $(2k, x + 2z)$.*

Proof. To analyze possible paths from $(0, x)$ to $(2k, x + 2z)$, we will find all possible values of each data coordinate in the node descriptions on that path. On a path from $(0, x)$ to $(2k, x + 2z)$, we change the i-th data coordinate twice: when we go from layer $i - 1$ to i and when we go from layer $k + i - 1$ to layer $k + i$. Thus, the only possible values of that coordinate are the initial value x_i, the final value $x_i + 2z_i$ and the intermediate value, say, $x_i + u$. Let $v = 2z_i - u$. Then $u, v \in F_{r,d}$ and $(u + v)/2 = z_i$. Because z_i is a corner (an extremal point) of the convex hull of $C_{r,d}$, this implies $u = v = z_i$. Therefore, there is only one possible value of each data coordinate for each node on such a path. ◀

We will call unique paths from $(0, x)$ to $(2k, x + 2z)$, described in the lemma, *straight paths*.

Consider a 2-TC-spanner H. We say that an edge $(u, v) \in H$ is *long* if the difference between the level of v and the level of u is at least k. For every straight path P, say, starting

at some $(0, x)$ and ending at some $(2k, x + 2z)$, there exists some (i, y) such that H contains edges $((0, x), (i, y))$ and $((i, y), (2k, x + 2z))$. We say that the path P is *covered* by these two edges. One of the two edges must be long, and thus it uniquely determines x and z. That is, a long edge can be used to cover at most one straight path. Therefore, H has to contain at least as many long edges as there are straight paths.

Now we give bounds on the numbers $n, |G|$ and s, where n is the number of nodes and s is the number of straight paths in G. Recall that $S_{2k}(G) = |G|$ and that s is a lower bound on $S_2(G)$. We use the following tight bound on the size of $F_{r,d}$ by Bárány and Larman [4].

▶ **Theorem 4.4.** *For every $d \geq 2$ there exist constants $c_1(d), c_2(d)$ such that*

$$c_1(d) r^{d\frac{d-1}{d+1}} \leq |F_{r,d}| \leq c_2(d) r^{d\frac{d-1}{d+1}}.$$

Let $c = |C_{r,d}|$, $a = |C_{3r,d}|/|C_{r,d}|$ and $f = |F_{r,d}|$.

Note that each layer of G has more nodes than the previous. In particular, for sufficiently large r, the size of layer $2k$ is at least $(3/2)^d$ times larger then the size of layer $2k-1$ because the range of the last data coordinate is expanded from $C_{2r,d}$ to $C_{3r,d}$, while the range of the remaining data coordinates remains the same. Therefore, n equals the size of layer $2k$ times a small constant factor, *i.e.,* $n \approx (ac)^k$. Since the out-degree of each node in G is f or 0, $|G| < nf < nac \approx n^{1+1/k}$.

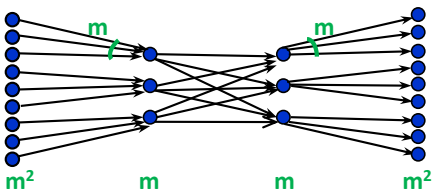
The number of straight paths is $s = (cf)^k$ because a straight path can be specified using any node in layer 0 (c^k choices) and any sequence of the first k edges of the path (f choices for each edge). We will show how to choose r and d, so that $s \geq n^{2-\epsilon}$ or, equivalently, $(cf)^k \geq (ac)^{k(2-\epsilon)}$. It is equivalent to $cf \geq (ac)^{2-\epsilon}$, and finally to $f \geq a^{2-\epsilon} c^{1-\epsilon}$. Since a does not increase with r while c does, we can guarantee $a^2 < c^{\epsilon/2}$ by choosing sufficiently large r. By choosing $d > 4/\epsilon$, we get $f > c^{1-\epsilon/2}$ (by Theorem 4.4). Thus, for sufficiently large r and d, $s \geq n^{2-\epsilon}$.

To summarize, $S_2(G)/S_{2k}(G) \geq s/|G| \geq n^{2-\epsilon}/n^{1+1/k} = n^{1-1/k-\epsilon}$, as stated. ◀

For stretch $k = 3$ (and hence for stretch $k = 4$), a simple construction yields a stronger (tight) lower bound.

▶ **Lemma 4.5.** *There exists an infinite family of graphs G on n nodes with $S_2(G)/S_3(G) = \Omega(\sqrt{n})$.*

Proof. For each m, we construct a 4-layered graph G with m^2 nodes in layers 1 and 4 and m nodes in layers 2 and 3, where the edges are directed from smaller to larger layers and are formed as follows. There is a complete bipartite graph between layers 2 and 3. Each node in layer 2 is connected to m nodes in layer 1, and each node in layer 1 has outdegree 1. The edges between layers 3 and 4 are constructed in the same manner. The resulting graph has $3m^2$ edges and is a 3-TC-spanner. A 2-TC-spanner of this graph must connect m^4 pairs of vertices in layers 1 and 4 via paths of length at most 2. Each shortcut edge can be used by at most m such pairs. Therefore, at least m^3 shortcuts are required. Setting $n = 2m^2 + 2m$, we obtain a graph with a 3-TC-spanner of size $O(n)$, for which every 2-TC-spanner requires $\Omega(n^{3/2})$ edges. ◀



5 Approximation Algorithm for k-TC-Spanner for large k

In this section, we present an algorithm for k-TC-Spanner that provides a better ratio than the algorithm from Theorem 2.1 for stretch $k = \Omega(\log n / \operatorname{loglog} n)$.

▶ **Theorem 5.1.** *k-TC-Spanner can be approximated with ratio $O(n/k^2)$ in polynomial time.*

$O(n/k^2)$-Approximation Algorithm for k-TC-Spanner
Input: directed graph $G = (V, E)$, desired stretch k
1. Let H be a transitive reduction of G.
2. While H contains vertices u, v such that $dist_H(u, v) > k$
   a. Let $(v_1 = u, v_2, ..., v_t = v)$ be the shortest path from u to v in H.
   b. Add a shortcut edge $(v_{\lfloor k/4 \rfloor}, v_{t-\lfloor k/4 \rfloor})$ to H.
3. Output H.

Proof. To prove that the above algorithm has the desired approximation ratio, it is enough to analyze it on each weakly connected component. Let OPT be the size of the sparsest k-TC-spanner on a weakly connected component with n nodes. Then $OPT \geq TR(G) \geq n-1$ where $TR(G)$ denotes the size of the transitive reduction of G. We will show that $O(n^2/k^2)$ shortcut edges are added to H, giving the desired ratio.

Observe that each shortcut edge decreases the distance from at least $k/2$ to less than $k/2$ for $\Omega(k^2)$ pairs of vertices (v_i, v_j) on the shortest path from u to v: namely, all pairs for which $i \leq \lfloor k/4 \rfloor$ and $j \geq t - \lfloor k/4 \rfloor$. Since initially there are $O(n^2)$ pairs of vertices at distance larger than $k/2$, only $O(n^2/k^2)$ shortcut with this property can be added. This completes the proof of the lemma. ◀

References

1 Alfred V. Aho, M. R. Garey, and Jeffrey D. Ullman. The transitive reduction of a directed graph. *SIAM J. Comput.*, 1(2):131–137, 1972.
2 Ingo Althöfer, Gautam Das, David Dobkin, Deborah Joseph, and José Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, 9(1):81–100, 1993.
3 Baruch Awerbuch. Communication-time trade-offs in network synchronization. In *PODC*, pages 272–276, 1985.
4 Imre Bárány and David Larman. The convex hull of the integer points in a large ball. *Mathematische Annalen*, 312:167–181, 1998.
5 Surender Baswana and Sandeep Sen. Approximate distance oracles for unweighted graphs in expected $\tilde{O}(n^2)$ time. *ACM Transactions on Algorithms*, 2(4):557–577, 2006.
6 Arnab Bhattacharyya, Elena Grigorescu, Madhav Jha, Kyomin Jung, Sofya Raskhodnikova, and David Woodruff. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. In *RANDOM*, pages 448–461, 2010.
7 Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David Woodruff. Transitive-closure spanners. In *SODA*, pages 932–941, 2009.
8 Arnab Bhattacharyya, Elena Grigorescu, Kyomin Jung, Sofya Raskhodnikova, and David Woodruff. Transitive-closure spanners of the hypercube and the hypergrid. ECCC Report TR09-046, 2009.
9 Arnab Bhattacharyya, Elena Grigorescu, Sofya Raskhodnikova, and David Woodruff. Steiner transitive-closure spanners of d-dimensional posets. Manuscript, 2010.
10 Hanls L. Bodlaender, Gerard Tel, and Nicola Santoro. Tradeoffs in non-reversing diameter. *Nordic Journal of Computing*, 1(1):111 – 134, 1994.

11 Edith Cohen. Fast algorithms for constructing t-spanners and paths with stretch t. *SIAM J. Comput.*, 28(1):210–236, 1998.

12 Edith Cohen. Polylog-time and near-linear work approximation scheme for undirected shortest paths. *JACM*, 47(1):132–166, 2000.

13 Lenore Cowen. Compact routing with minimum stretch. *J. Algorithms*, 38(1):170–183, 2001.

14 Lenore Cowen and Christopher G. Wagner. Compact roundtrip routing in directed networks. *J. Algorithms*, 50(1):79–95, 2004.

15 M. Elkin. Computing almost shortest paths. In *PODC*, pages 53–62, 2001.

16 Michael Elkin and David Peleg. Strong inapproximability of the basic k-spanner problem. In *ICALP*, pages 636–647, 2000.

17 Michael Elkin and David Peleg. The client-server 2-spanner problem with applications to network design. In *SIROCCO*, pages 117–132, 2001.

18 Michael Elkin and David Peleg. The hardness of approximating spanner problems. *Theory Comput. Syst.*, 41(4):691–729, 2007.

19 William Hesse. Directed graphs requiring large numbers of shortcuts. In *SODA*, pages 665–669, 2003.

20 D. Hochbaum, editor. *Approximation Algorithms for NP-hard Problems*. PWS Publishing Company, Boston, 1997.

21 G. Kortsarz and D. Peleg. Generating sparse 2-spanners. *J. Algorithms*, 17(2):222–236, 1994.

22 Guy Kortsarz. On the hardness of approximating spanners. *Algorithmica*, 30(3):432–450, 2001.

23 David Peleg. *Distributed computing: a locality-sensitive approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

24 David Peleg and Alejandro A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13(1):99–116, 1989.

25 David Peleg and Jeffrey D. Ullman. An optimal synchronizer for the hypercube. *SIAM J. Comput.*, 18(4):740–747, 1989.

26 David Peleg and Eli Upfal. A trade-off between space and efficiency for routing tables. *JACM*, 36(3):510–530, 1989.

27 Sofya Raskhodnikova. Transitive-closure spanners: a survey. In Oded Goldreich, editor, *Property Testing*, volume LNCS 6390, pages 167–196. Springer, Heidelberg, 2010.

28 Mikkel Thorup. On shortcutting digraphs. In *Graph-Theoretic Concepts in Computer Science*, volume 657, pages 205–211, 1993.

29 Mikkel Thorup. Shortcutting planar digraphs. *Combinatorics, Probability & Computing*, 4:287–315, 1995.

30 Mikkel Thorup. Parallel shortcutting of rooted trees. *J. Algorithms*, 23(1):139–159, 1997.

31 Mikkel Thorup and Uri Zwick. Compact routing schemes. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 1–10, 2001.

32 Mikkel Thorup and Uri Zwick. Approximate distance oracles. *JACM*, 52(1):1–24, 2005.

33 Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, New York, NY, USA, 2007.

Combinatorial Problems with Discounted Price Functions in Multi-agent Systems

Gagan Goel[1], Pushkar Tripathi[1], and Lei Wang[1]

1 **College of Computing**
Georgia Institute of Technology
Atlanta, USA
gagan,pushkar.tripathi,lwang@cc.gatech.edu

Abstract

Motivated by economic thought, a recent research agenda has suggested the algorithmic study of combinatorial optimization problems under functions which satisfy the property of decreasing marginal cost. A natural first step to model such functions is to consider submodular functions. However, many fundamental problems have turned out to be extremely hard to approximate under general submodular functions, thus indicating the need for a systematic study of subclasses of submodular functions that are practically motivated and yield good approximation ratios. In this paper, we introduce and study an important subclass of submodular functions, which we call *discounted price functions*. These functions are succinctly representable and generalize linear(additive) price functions. We study the following fundamental combinatorial optimization problems: edge cover, spanning tree, perfect matching and $s-t$ path. We give both upper and lower bound for the approximability of these problems.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.436

1 Introduction

In the algorithmic theory of combinatorial optimization, much of the attention has been focused on problems where one wishes to optimize an *additive* function under some combinatorial constraints. In these problems we are given a ground set E of elements and a collection Ω of subsets of the ground set ($\Omega \subseteq 2^E$) that is usually defined implicitly by some combinatorial property (such as the set of all spanning trees in a graph). We are also given a cost for each element in E and the price of a subset $S \in \Omega$ is defined to be the sum of costs of the elements in S. The objective is to find a subset in Ω with a minimum price.

However, additive price functions do not always model the complex dependencies of the prices in a real-world setting. It is widely believed in economics that price structure satisfies the decreasing marginal cost property. Intuitevly, it says that the price of adding an element to a larger set is less than adding it to a smaller set. Largely motivated by this, in recent years, a research agenda that has emerged out [9, 10, 11, 12, 20] is to study combinatorial optimization problems under *submodular* functions. Submodular functions form a very broad class of functions and mathematically capture the property of decreasing marginal cost. A function $f : 2^E \rightarrow R^+$ is said to be submodular if and only if for any two subsets S and $T \subseteq E$ such that $S \subseteq T$, the following holds: $f(T \cup i) - f(T) \leq f(S \cup i) - f(S)$ for all elements $i \in E - T$.

Unfortunately, many of the fundamental optimization problems have turned out to be extremely hard under submodular functions [9, 10, 11, 12, 20]. Thus, from a practical standpoint, the applicability of these results is not very well-founded as the class of submodular functions might be much more general than real-world functions. Moreover, the class of submodular functions is defined over an exponentially large domain and thus requires

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 436–446

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

exponential time to write down the function explicitly. This may not be the case in real-world applications.

In this paper, we wish to explore functions that lie between the additive functions and the general submodular functions, and that are also succinctly representable. In particular, we study *discounted price functions* in which we are given an additive function c and a discount function $d : R^+ \to R^+$ that is a concave curve. The price of any subset S is defined to be $d(c(S))$. It is not difficult to see that dicounted price functions form a subclass of submodular functions. Discounted price functions have strong theoretical motivation as well. A common technique (due to [10]) for designing optimal algorithms under general submodular functions is to first approximate submodular functions by *ellipsoid* functions. These ellipsoid functions form a special class of discounted price functions.

We study discounted price functions in a multi-agent setting. This is motivated by the observation that often in a real-world scenario there are multiple agents, with different price functions, each of whom can build different parts of the required combinatorial structure. For instance, in the case of spanning tree, it might be more cost effective to buy only a subset of the edges of the final tree from a particular agent. For additive functions, it is easy to see that having multiple agents doesn't change the complexity of the original problem. However, this is not the case for more general price functions.

Discounted Price Model

We define a function $d : \mathrm{R}^+ \to \mathrm{R}^+$ to be a discounted price function if it satisfies the following properties: (1) $d(0) = 0$; (2) d is increasing; (3) $d(x) \leq x$ for all x; (4) d is concave.

We study combinatorial problems in the following general setting. We are given a set of elements E, and a collection Ω of its subsets. We are also given a set $\mathcal{A}$ of k agents where each agent $a \in \mathcal{A}$ specifies a cost function $c_a : E \to \mathrm{R}^+$ where $c_a(e)$ indicates her cost for the element e. Each agent also declares a discounted price function d_a. If an agent a is assigned a set T of elements, then her total price is specified by $d_a(\sum_{e \in T} c_a(e))$. This is called her *discounted price*. For the ease of notation we will use $d_a(T)$ to denote $d_a(\sum_{e \in T} c_a(e))$. The objective is to select a subset S from Ω and a partition $S_1, S_2, ..., S_k$ of S, such that $\sum_{a \in \mathcal{A}} d_a(S_a)$ is minimized.

We study the following four problems over an undirected graph $G = (V, E)$.

- **Discounted Edge Cover**: In this problem, Ω is chosen to be the collection of edge covers.
- **Discounted Spanning Tree**: In this problem, Ω is the collection of spanning trees of the graph.
- **Discounted Perfect Matching**: In this problem, we assume the graph has an even number of vertices. Ω is chosen to be the collection of perfect matchings of the graph.
- **Discounted $s - t$ Path**: In this problem, we are given two fixed vertices s and t. Ω consists of all paths connecting s and t.

Our Results

In section 2.1, we show that the discounted edge cover and spanning tree problems are hard to approximate within a factor of $(1 - o(1)) \log n$ unless P = NP. In section 2.2 and 2.3, we amplify this result to show that $s - t$ path and matching are hard to approximate within any polylog factor.

On the algorithmic front, in section 3.1 and 3.2, we show that our results are tight by giving $\log n$-approximate algorithms for the discounted edge cover and spanning tree

problems. In section 3.3, we describe simple $O(n)$-approximate algorithms for discounted $s-t$ and perfect matching. We leave the design of sublinear approximation algorithms for these two problems as an open question.

Related work

The classical versions of edge cover, spanning tree, perfect matching and $s-t$ path are well studied and polynomial time algorithms are known for all these problems, see [18, 14, 3, 7, 6, 4]. These have served as part of the most fundamental combinatorial optimization models in the development of computer science and operations research.

Recently, Svitkina and Fleischer [20] generalized the linear(additive) cost settings of some combinatorial optimization problems such as sparsest cut, load balancing and knapsack to a submodular cost setting. They gave $\sqrt{n/\log n}$ upper and lower bounds for all these problems. [11] also studied combinatorial optimization problems in the single agent setting.

Multi-agent setting was first introduced in [9]. In their model of *combinatorial problems with multi-agent submodular cost functions* they gave $\Omega(n)$ lower bounds for the problems of submodular spanning tree and perfect matching, and gave a $\Omega(n^{2/3})$ lower bound for submodular $s-t$ path where the submodular functions are given by value oracles. They also gave the matching upper bounds for all these problems. We remark that the lower bounds presented in [20, 9] are information theoretic and not computational.

Similar generalization from additive objective functions to the more general submodular functions was applied to maximization problems in [8, 15, 2]. The multi-agent generalization of maximization problems which corresponds to *combinatorial auction* has been extensively studied both in computer science and economics [5, 17] and tight information theoretic lower bounds are known for these problems, see [16].

2 Hardness of Approximation

In this section we present hardness of approximation results for the four problems defined earlier. Unlike some of the previous work on combinatorial optimization [9, 20] over non-linear cost functions, the bounds presented here are not information theoretic but are contingent on $\mathrm{P} \neq \mathrm{NP}$. In section 2.1, we show that all our problems are hard to approximate within factor $\log n$. In section 2.2 and 2.3, we amplify the hardness of approximation for discounted $s-t$ path and perfect matching to $O(\log^c n)$ for any constant c.

Recall that in each of the problems we are given a graph $G=(V,E)$ over n vertices. We are also given a set $\mathcal{A}$ of k agents each of whom specifies a cost $c_a : E \to \mathrm{R}^+$. Here $c_a(e)$ is the cost for building edge e for agent a. Each agent also specifies a discounted price function given by $d_a : \mathrm{R}^+ \to \mathrm{R}^+$. The objective is to build a specified combinatorial structure using the edges in E, and allocate these edges among the agents such that the sum of discounted prices for the agents is minimized.

2.1 Basic Reduction

To show the logarithmic hardness of approximation for the problems stated earlier we consider the following general problem and use a reduction from set cover to establish its hardness.

Discounted Reverse Auction: We are given a a set E of n items and a set $\mathcal{A}$ of agents each of whom specifies a function $c_a : E \to \mathrm{R}^+$. Here $c_a(e)$ is the cost for procuring item e from agent a. Each agent also specifies a discounted price function given by $d_a : \mathrm{R}^+ \to \mathrm{R}^+$.

The task is to find a partition $\mathcal{P} = \{P_1 \cdots P_k\}$ of E such that $\sum_{a \in \mathcal{A}} d_a(\sum_{e \in P_a} c_a(e))$ is minimized.

▶ **Lemma 1.** *It is hard to approximate the discounted reverse auction problem within factor* $(1 - o(1)) \log n$ *unless* P = NP.

Proof. We reduce set cover to the discounted reverse auction problem to prove this result. Consider an instance $\mathcal{I} = (U, C, w)$ of set cover where we wish to cover all elements in the universe U using sets from C and minimize the sum of weights under the weight function $w : C \to \mathrm{R}^+$. We define an instance, $\mathcal{I}'$ of our discounted reverse auction problem corresponding to $\mathcal{I}$ in the following way. Let U be the set of items. For every set $S \in C$ define an agent a_S, whose cost function c_a assigns the value $w(S)$ for every element $s \in S$ and sets the cost of all other elements in U to be infinity. The discounted price function for the agent is shown in figure 1. Here the slope of the second segment is small enough.

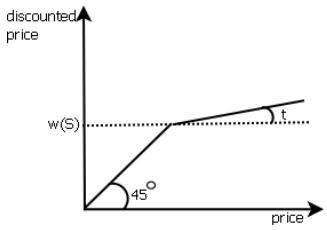


Figure 1 Discount function for agent corresponding to set S

Consider a solution for $\mathcal{I}'$ where we procure at least one item from agent a_S; then we can buy all elements in S from a_S without a significant increase in our payment. So the cost of the optimal solution to $\mathcal{I}$ can be as close to the price of the optimal solution for $\mathcal{I}'$ as we want. By [1, 19], set cover is hard to approximate beyond a factor of $\log n$ unless P = NP. Therefore the discounted reverse auction problem can not be approximated within factor $(1 - o(1)) \log n$ unless P = NP. ◀

This reduction can be extended to other combinatorial problems in this setting to give logarithmic hardness of approximation for many combinatorial problems. This can be achieved by considering an instance of the problem where we have just one combinatorial object and our task is to allocate it optimally among the agents. For example, for the discounted spanning tree problem we consider the instance when the input graph is itself a tree and we have to optimally allocate its edges among the agents to minimize the total price. Thus, we have the following:

▶ **Theorem 2.** *It is hard to approximate discounted edge cover, spanning tree, perfect matching and* $s - t$ *path within factor of* $(1 - o(1)) \log n$ *on a graph with* n *vertices unless* P = NP.

2.2 Amplification: Hardness for Discounted $s - t$ Path

In this section we consider the discounted $s - t$ path problem between two given vertices s and t. We show that unless P = NP, this problem is hard to approximate within a factor of $O(\log^c n)$ for any fixed constant c. The proof is based on amplification of the result of theorem 2. This is done by repeatedly applying a transformation σ on the given family of problem instances, which amplifies the approximation factor on every application. Each

application of σ also increases the size of the graph but only by a polynomial(in n) factor. We now describe the transformation formally.

Consider the following instance $(G, \mathcal{A}, \mathcal{U})$: We are given a graph $G = (V, E)$, vertices s,t and a set $\mathcal{A}$ of agents. We are also given a collection $\mathcal{U} = \{U_a\}_{a \in \mathcal{A}}$. Here $U_a \subseteq E$ specifies the set of edges that can be assigned to agent a, i.e., $c_a(e) = 1$ for all $e \in U_a$ and $c_a(e) = +\infty$ otherwise. The discounted price function d_a is such that $d_a(x) = x$ for all $x \leq 1$ and 1 for all $1 < x < +\infty$. Observe that under this assumption, for any set S of edges, $d_a(S)$ has value 1 if $S \subseteq U_a$ and ∞ otherwise. We may assume that the sets U_a for $a \in \mathcal{A}$ are pairwise disjoint, by replacing a single edge that can be assigned to multiple agents by parallel edges and assigning them to each of the agents. In future discussion, we will use $\mathcal{F}$ to denote the family of instances $\{(G, \mathcal{A}, \mathcal{U})\}$.

We define the transformation $\sigma : \mathcal{F} \to \mathcal{F}$ that takes an instance $I = (G, \mathcal{A}, \mathcal{U})$ in $\mathcal{F}$, and generates another instance $I^{\otimes} = (G^{\otimes}, \mathcal{A} \times \mathcal{A}, \mathcal{U}^{\otimes})$ as follows. The graph $G^{\otimes} = (V^{\otimes}, E^{\otimes})$ is constructed from G by replacing each edge $(u, v) \in E$ with a copy of the graph G such that s coincides with u and t coincides with v. Thus any edge $e \in E$ can be identified with a subgraph G^e, of $G^{\otimes}$ that is isomorphic to G. Each $e' \in G^e$ has a cost $c_a(e)$ for each agent a and we define $\rho(e') = e$ and define $\gamma(e')$ to be the edge corresponding to e' in G under this isomorphism. There are $|\mathcal{A}|^2$ agents in the new instance who are indexed by $\mathcal{A} \times \mathcal{A}$. We define the elements of $\mathcal{U}^{\otimes}$ as $U^{\otimes}_{(a_1,a_2)} = \{e' \mid \rho(e') \in U_{a_1} \text{ and } \gamma(e') \in U_{a_2}\}$.

Note that $|E^{\otimes}| = |E|^2$, i.e. the size of instance $I^{\otimes}$ is bounded by a polynomial in the size of I. We define $\sigma(\mathcal{F}) = \{\sigma(I) \mid \forall I \in \mathcal{F}\}$. In lemma 3, we show that we can amplify that hardness result from theorem 2 by applying the transformation σ repeatedly.

▶ **Lemma 3.** *If $\mathcal{H} = \sigma^r(\mathcal{F})$ is a family of instances for the $s-t$ path problem that is hard to approximate to a factor better than α, then $\sigma(\mathcal{H})$ is hard to approximate within a factor $O(\alpha^2)$.*

Proof. Let $I = (G, s, t, \mathcal{A}, \mathcal{U})$ be an instance in $\mathcal{H}$. Let us begin by making some observations about the structure of an optimal solution for $\sigma(I) = (G^{\otimes}, \mathcal{A}, \mathcal{U}^{\otimes})$.

▶ Claim 1. If there is a $s-t$ path of price β in G, then there is a $s-t$ path in $G^{\otimes}$ of price at most β^2.

Proof of claim 1. Let $P = e_1, e_2 \dots e_t$ be a path of price β in G. We can construct a $s-t$ path in $G^{\otimes}$ by considering the set of graphs $G^{e_1} \dots G^{e_t}$ and picking the edges corresponding to the edges in P in each of these copies. It can be verified that this gives us a valid path that has price β^2. ◀

Next we note that the converse is also true.

▶ Claim 2. If there is a $s-t$ path of price β^2 in $G^{\otimes}$ then there is a $s-t$ path in G of price at most β.

Proof of claim 2. Let P be a path of price β^2 in $G^{\otimes}$. Let $G^{e_1} \dots G^{e_t}$ be the copies of G that have non-empty intersection with P. Two cases may arise. Either the set of edges $\{e_1 \dots e_t\}$ belong to at most β distinct agents in $\mathcal{A}$ or they belong to more than β agents in $\mathcal{A}$. Note that the set of edges $\{e_1 \dots e_t\}$ form a path in G, and in the first case this path has price at most β. In the second case, the price of edges in $P \cap G^{e_i}$ must be less than β for some copy G^{e_i} of graph G. These edges also form a $s-t$ path in G of price at most β. Thus in both cases we can find a path of price at most β in G. ◀

Using the observations above, if the price of the optimal solution to I is OPT, then the price of the optimal solution to $\sigma(I)$ is OPT^2. Furthermore, if we can approximate the optimal solution to $\sigma(I)$ to within a factor of $o(\alpha^2)$ then we can approximate the optimal solution for I to better than $o(\alpha)$, using the construction in claim 2. This yields the desired contradiction. ◀

By theorem 2, $\mathcal{F}$ is hard to approximate within a factor of $\log n$. Using this as the base case and applying lemma 3 repeatedly we have the following theorem.

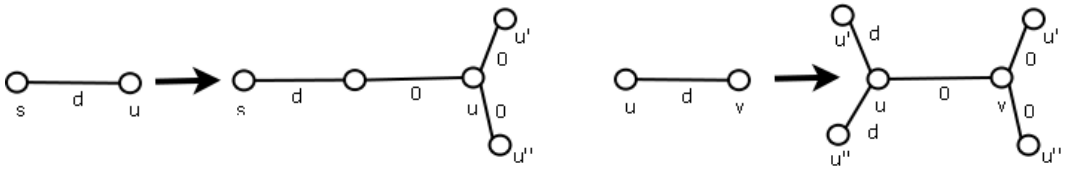
▶ **Theorem 4.** *The discounted shortest $s-t$ path problem is hard to approximate within a factor of $O(\log^c n)$ for any fixed constant $c > 0$.*

2.3 Reduction: Hardness for Discounted Perfect Matching

In this section we consider the discounted perfect matching problem. We show that unless P = NP, this problem is hard to approximate within a factor of $O(\log^c n)$ for any fixed constant c. The proof is based on a factor preserving reduction from the $s-t$ path problem. We now describe our reduction:

▶ **Lemma 5.** *Let A be a β-approximate algorithm for the perfect matching problem, then we can get a β-approximation for the $s-t$ path problem using A as a subroutine.*

Proof. Suppose we are given a graph $G = (V, E)$. Construct an auxiliary graph G^* in the following way: Replace every vertex $v \in V$ by v' and v'' and add an edge connecting them. The price of this edge is zero for every agent. We replace each edge $uv \in E$ with the gadgets shown in figure 2.



Gadgets for edges incident on s or t

Gadgets for the edges not incident on s or t

Figure 2 Gadgets

On this graph G^*, use the algorithm A to get the minimum weight matching. Let M be the matching returned. We can interpret M as a $s-t$ path in G in the following way. Let $g(uv)$ be the edges in G^* corresponding to the edge uv for the gadget shown in figure 2. Observe that either one or two edges of every such gadget must belong to M. Let S be the set of edges in G such that two edges in their corresponding gadget belong to M. One can check that every vertex in V is incident with zero or two edges from S, whereas s and t are each incident with exactly one edge in S. Therefore S consists of an $s-t$ path P_S and some other circuits. Now the circuits in S must have cost zero. This is because if a circuit has positive cost then the cost of the matching can be reduced further by pairing up the vertices in the circuit as shown in figure 3.

◀

Note that the reduction defined in lemma 5 defines a cost preserving bijection between $s-t$ paths in G to perfect matchings in G^*. Thus, using theorem 4 we have :

▶ **Theorem 6.** *The discounted perfect matching problem is hard to approximate within a factor of $O(\log^c n)$ for any fixed constant $c > 0$.*

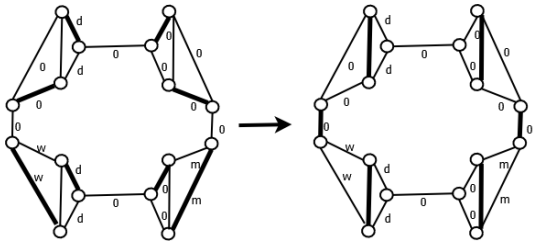


Figure 3 Circuits not involving edges in S should have zero costs

3 Algorithms for Discounted Combinatorial Optimization

In this section, we present approximation algorithms for the four problems defined earlier.

3.1 Discounted Edge Cover

We will establish a factor $O(\log n)$ algorithm for the discounted edge cover problem.

Given a discounted edge cover instance, we construct a set cover instance such that: 1) An optimal edge cover corresponds to a set cover with the same cost and 2) A set cover corresponds to an edge cover with a smaller price. For the set cover instance, we apply the greedy algorithm from [13] to get a set cover whose cost is within $\log n$ of the optimal cost. The corresponding edge cover gives a $\log n$ approximation of the optimum edge cover. We remark that we will have exponentially many sets in the set cover instance that we construct for our problem. To apply the greedy algorithm, we need to show that in each step, the set with the lowest *average cost* can be found in polynomial time.

Now we state our algorithm formally. Consider a set cover instance where we have to cover the set of vertices, V, with $k2^n$ subsets which are indexed by $(a, S) \in \mathcal{A} \times 2^V$. The cost of the set (a, S), denoted by $cost(a, S)$, is defined as the minimum discounted price of an edge cover for the vertices in S that can be built by agent a. For the instance of set cover described above, we apply the greedy algorithm [13] to get a set cover $\mathcal{S}$. Let U_a be the set of vertices covered by sets of the form $(a, S) \in \mathcal{S}$. Let C_a be agent a's optimal discounted edge cover of the vertices in U_a. We output $\{C_a : a \in \mathcal{A}\}$ as our solution.

The correctness of the algorithm follows from the observation that each C_a is a cover of U_a thus their union must form an edge cover for V.

Now we show that the running time of the algorithm is polynomial in k and n. Given U_a, C_a can be found in polynomial time. Thus our algorithm can be implemented in polynomial time, if we can implement the greedy algorithm on our set cover instance efficiently.

Recall that the greedy algorithm from [13] covers the ground set iteratively. Let Q be the set of covered elements at the beginning of a phase. The *average cost* of a set (a, S) is defined as $\alpha_a(S) = cost(a, S)/|S - Q|$. In every iteration the algorithm picks the set with the smallest average cost until all the vertices are covered. To show that this algorithm can be implemented efficiently, we only need to show the following lemma.

▶ **Lemma 7.** *For any $Q \subset V$, we can find $min\{cost(a, S)/|S - Q| : (a, S) \in \mathcal{A} \times 2^V\}$ in polynomial time.*

Proof. We can iterate over all choices of agent $a \in \mathcal{A}$, thus the problem boils down to finding $\min\{cost(a, S)/|S - Q| : S \subseteq V\}$ for each $a \in \mathcal{A}$.

For each integer d, if we can find $\min\{cost(a, S) : |S - Q| = d\}$ in polynomial time, then we are done since then we can just search over all the possible sizes of $S - Q$. Unfortunately,

it is NP-hard to compute $\min\{cost(a,S) : |S-Q| = d\}$ for all integer d. We will use claim 3 to circumvent this problem.

▶ Claim 3. For any graph $G = (V,E)$ and $Q \subseteq V$ and for any positive integer d, we can find the set (a,S) minimizing $cost(a,S)$ such that $|S-Q|$ is at least d, in polynomial time.

Proof. To find the desired set we construct a graph $G' = (V',E')$ as follows: Add a set $X \cup Y$ to the set of vertices in G, where $|X| = |Q|$ and $|Y| = |V| - |Q| - d$. Match every vertex in X to a vertex in Q with an edge of cost 0. Connect each vertex in Y to each vertex in V by an edge of very large cost. Set the cost of each edge $e \in E$ as $c_a(e)$. Find the minimum cost edge cover in G'. Let S^* be the set of vertices not adjacent to $X \cup Y$ in such a cover. It is easy to verify that S^* is the desired set. ◀

By claim 3 above we can generate a collection of subsets $\{S_i \subseteq V : 1 \leq i \leq n\}$, such that (a,S_i) has the lowest value of $cost(a,S)$ among all sets S which satisfy $|S-Q| \geq i$.

▶ Claim 4. $\min\{cost(a,S)/|S-Q| : S \subseteq V\} = min\{cost(a,S_i)/|S_i - Q| : 1 \leq i \leq n\}$.

Proof. Let S^* be the set that has the minimum average cost with respect to agent a. Suppose $|S^* - Q| = d$. By our choice of S_d, we have $|S_d - Q| \geq d = |S^* - Q|$ and $cost(a,S_d) \leq cost(a,S^*)$. Therefore we have $cost(a,S_d)/|S_d - Q| \leq cost(a,S^*)/|S^* - Q|$, hence they must be equal. ◀

By iterating over all $a \in \mathcal{A}$, we can find $\min\{cost(a,S)/|S-Q| : (a,S) \in \mathcal{A} \times 2^V\}$ in polynomial time. ◀

Next we show that the approximation factor of our algorithm is $\log n$. Let OPT_{EC} and OPT_S denote the costs of the optimal solutions for the discounted edge cover instance and the corresponding set cover instance respectively. Let $\{C_a : a \in \mathcal{A}\}$ be the edge cover reported by our algorithm. For all $a \in \mathcal{A}$ let O_a be the set of vertices covered by agent a in the optimal edge cover. The sets $\{(a,O^a) : a \in \mathcal{A}\}$ form a solution for the set cover instance. Therefore $OPT_S \leq OPT_{EC}$.

Since we use the greedy set cover algorithm to approximate OPT_S, we have

$$\sum_{a \in \mathcal{A}} d_a(C_a) \leq (\log n) OPT_S \leq (\log n) OPT_{EC}$$

Thus we have the following theorem.

▶ **Theorem 8.** *There is a polynomial time algorithm which finds a $O(\log n)$-approximate solution to the discounted edge cover problem for any graph over n vertices.*

3.2 Discounted Spanning Tree

In this section, we study the discounted spanning tree problem and establish an $O(\log n)$ approximation algorithm for this problem.

Let us first consider a simple $O(\log^2 n)$-approximation algorithm. Observe that a spanning tree is an edge cover with the connectivity requirement. If we apply the greedy edge cover algorithm in section 3.1, there is no guarantee that we will end up with a connected edge cover. We may get a collection of connected components. We can subsequently contract these components and run the greedy edge cover algorithm again on the contracted graph. We repeat this until there is only one connected component. By this method, we will get a connected edge cover containing a spanning tree.

Now we will analyze the above algorithm. Let OPT_{ST} be the price of the minimum discounted spanning tree. After each execution of the greedy edge cover algorithm, there is no isolated vertex, hence the contraction decreases the number of vertices by at least a a factor of half, therefore we will have to run the greedy edge cover algorithm at most $O(\log n)$ times. Let OPT^r_{EC} be the price of the minimum edge cover for the graph obtained after the r^{th} contraction and let C_r be the edge cover that we produce for this iteration. Using theorem 8, the price of C_r is at most $(\log n)OPT^r_{EC}$. It is easy to see that OPT^r_{EC} is at most OPT_{ST} for every r. Hence the price of C_r is bounded by $\log n \cdot OPT_{ST}$. Since there are at most $O(\log n)$ iterations, the price of the spanning tree produced by the above algorithm is bounded by $O(\log^2 n)OPT_{ST}$.

We observe that two main steps in the above algorithm are greedy edge cover and contraction. Intuitively, they are used to satisfy the covering and connectivity requirements respectively. The algorithm proceeds by alternately invoking these subroutines. Based on this observation, our idea to get an $O(\log n)$ approximation algorithm is to apply the following greedy algorithm: rather than apply contraction after each complete execution of the greedy edge cover, we interleave contraction with the iterations of the greedy edge cover algorithm. After each iteration, we modify the graph to coerce our algorithm to get a connected edge cover at the end.

Now we describe our greedy algorithm. For every agent a and subset of vertices S we define $cost(a, S)$ as the cost of the optimal edge cover for S. We define the *average cost* of a set (a, S) as $\alpha^S_a = cost(a, S)/|S|$. The algorithm proceeds in phases and each phase has two steps, **search** and **contraction**. In the r^{th} phase, during the search step we find the set (a_r, S_r) with the lowest average cost and set the *potential* of each vertex $v \in S_r$ as $p(v) = \alpha^{S_r}_{a_r}$. The search step is followed by a contraction step, where we modify the graph by contracting every connected component in the induced subgraph of agent a_r's optimal edge cover for the set S_r. After this we begin the next phase. The algorithm terminates when we have contracted the original graph to a single vertex. For every agent, we find the set of all edges assigned to her across all the search steps declare this as her bundle of assigned edges. Finally remove unnecessary edges from the set of assigned edges to get a spanning tree.

It is easy to see that we get a connected edge cover at the end of the algorithm, which proves the correctness of the algorithm. To analyze the running time, we observe that there can be at most n phases and by Lemma 7, each phase can be implemented in polynomial time. Hence the algorithm runs in polynomial time.

Next, we prove that the approximation factor of the algorithm is $O(\log n)$. Let OPT_{ST} be the price of the optimal solution of the discounted spanning tree instance and let $\{T_a : a \in \mathcal{A}\}$ be our solution. Let V' be the set of contracted vertices we produced during the algorithm. Number the elements of V and V' in the order in which they were covered by the algorithm, resolving ties arbitrarily. Suppose $V = \{v_1, ..., v_n\}$ and $V' = \{z_1, ..., z_{n'}\}$. Obviously, $n' \leq n$.

It is easy to verify that $\sum_a d_a(T_a) \leq \sum_i p(v_i) + \sum_j p(z_j)$. Therefore we only need to bound the potentials of the vertices in $V \cup V'$.

▶ Claim 5. $p(v_i) \leq \frac{OPT_{ST}}{n-i+1}$ for any $i \in \{1 \cdots n\}$ and $p(z_j) \leq \frac{OPT_{ST}}{n'-j}$ for any $j \in \{1 \cdots n'\}$.

Proof. For $i \in \{1 \cdots n\}$, suppose v_i is covered in phase r. Let G^r be the underlying graph at the beginning of phase r. Since $v_i, v_{i+1}, ..., v_n$ are not covered before phase r, G^r contains at least $n - i + 1$ vertices. Since the optimal spanning tree can cover the vertices in G^r by a price of OPT_{ST}, by our greedy choice, $p(v_i) \leq OPT_{ST}/(n - i + 1)$.

Similarly, let $1 \leq j \leq n'$ and assume z_j is covered in phase r. Since we should be able to produce $z_{j+1}, z_{j+2}, ..., z_{n'}$ from contraction on vertices of G^r, there are at least $n' - r$ vertices in G^r. Therefore we have $p(z_j) \leq OPT_{ST}/(n' - j)$. ◀

From the above claim, we have

$$\sum_a d_a(T_a) \leq \sum_{1\leq i\leq n} \frac{OPT_{ST}}{n-i+1} + \sum_{1\leq j\leq n'} \frac{OPT_{ST}}{n'-j} \leq (\log n + \log n')OPT_{ST} \leq O(\log n)OPT_{ST}$$

Therefore we have the following theorem:

▶ **Theorem 9.** *There is a polynomial time algorithm which finds an $O(\log n)$-approximate solution to the discounted spanning tree problem for any graph with n vertices.*

3.3 Discounted $s-t$ Path and Perfect Matching

In sections 2.2 and 2.3 we showed that unlike edge cover and spanning tree, no polylog-approximate algorithm is likely to exist for discounted $s-t$ path and perfect matching.

Now we describe a simple n-approximate algorithm for discounted $s-t$ path problem. For each edge e, define $w_e = \min_{a\in\mathcal{A}} d_a(c_a(e))$ and for each $s-t$ path P, define its weight $w(P) = \sum_{e\in P} w_e$. Use Dijkstra's algorithm to find a path P_0 with the minimum weight and output it as the solution. Allocate the edges in P_0 as follows: for each edge $e \in P$, we allocate e to the agent a such that $d_a(c_a(e)) = w_e$, with ties broken arbitrarily.

For the analysis, let us define S_a to be the set of edges allocated to agent a in our solution. Since d_a is concave, we have $d_a(S_a) \leq \sum_{e\in S_a} d_a(c_a(e))$. Therefore, the total price of our solution is bounded by $\sum_{a\in\mathcal{A}} \sum_{e\in S_a} d_a(c_a(e))$ which is exactly $w(P_0)$. Let OPT be the path chosen in the optimal solution (as an abuse of notation, we also use OPT to denote the optimal value) and OPT_a be the set of edges on the path allocated to agent a. By our choice of P_0 and weight w, we have $w(P_0) \leq w(OPT) = \sum_{a\in\mathcal{A}} \sum_{e\in OPT_a} w_e \leq \sum_{a\in\mathcal{A}} \sum_{e\in OPT_a} d_a(c_a(e))$. Since d_a is increasing, we have

$$\sum_{e\in OPT_a} d_a(c_a(e)) \leq |OPT_a| \cdot d_a(OPT_a) \leq n \cdot d_a(OPT_a).$$

Therefore $w(OPT) \leq n \sum_{a\in\mathcal{A}} d_a(OPT_a) = n \cdot OPT$. This implies that our algorithm is an n-approximate algorithm.

We apply the same idea for discounted perfect matching problem. Define the weight of a perfect matching M as $w(M) = \sum_{e\in M} w_e$. Use Edmond's algorithm to find a minimum weight perfect matching M_0 for this weight function. For every $e \in M_0$, allocate it to the agent a such that $c_a(e) = w_e$. By a similar argument as above, we can show that this is a n approximate algorithm.

Acknowledgements

We would like to thank Vijay Vazirani for is continued guidance and support throughout the development of this paper. Also, we would like to thank the reviewers for many useful suggestions.

References

1 Noga Alon, Dana Moshkovitz, and Shmuel Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. Algorithms*, 2(2):153–177, 2006.

2 Gruia Calinescu, Chandra Chekuri, Martin Pál, and Jan Vondrák. Maximizing a submodular set function subject to a matroid constraint (extended abstract). In *IPCO '07: Proceedings of the 12th international conference on Integer Programming and Combinatorial Optimization*, pages 182–196, Berlin, Heidelberg, 2007. Springer-Verlag.

3 William Cook and Andre Rohe. Computing minimum-weight perfect matchings. *INFORMS J. on Computing*, 11(2):138–148, 1999.

4 Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

5 Sven de Vries and Rakesh V. Vohra. Combinatorial auctions: A survey. *INFORMS Journal on Computing*, (3):284–309, 2003.

6 Edsger W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

7 Jack Edmonds. Paths,trees and flowers. *Canad J.Math*, 17, 1965.

8 Uriel Feige and Vahab S. Mirrokni. Maximizing non-monotone submodular functions. In *In Proceedings of 48th Annual IEEE Symposium on Foundations of Computer Science (FOCS*, page 2007, 2007.

9 Gagan Goel, Chinmay Karande, Pushkar Tripathi, and Lei Wang. Approximability of combinatorial problems with multi-agent submodular cost functions. In *FOCS '09: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009.

10 M.X. Goemans, N.J.A. Harvey, S. Iwata, and V. Mirrokni. Approximating submodular functions everywhere. In *SODA '09: Proceedings of the 20th ACM-SIAM Symposium on Discrete Algorithms*, 2009.

11 Satoru Iwata and Kiyohito Nagano. Submodular function minimization under covering constraints. In *FOCS '09: Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009.

12 Stefanie Jegelka and Jeff Bilmes. Cooperative cuts: Graph cuts with submodular edge weights. *Technical Report*, 2010.

13 David S. Johnson. Approximation algorithms for combinatorial problems. In *STOC '73: Proceedings of the fifth annual ACM symposium on Theory of computing*, pages 38–49, New York, NY, USA, 1973. ACM.

14 Joseph B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, February 1956.

15 Jon Lee, Vahab S. Mirrokni, Viswanath Nagarajan, and Maxim Sviridenko. Non-monotone submodular maximization under matroid and knapsack constraints. In *STOC '09: Proceedings of the 41st annual ACM symposium on Theory of computing*, pages 323–332, New York, NY, USA, 2009. ACM.

16 Vahab Mirrokni, Michael Schapira, and Jan Vondrak. Tight information-theoretic lower bounds for welfare maximization in combinatorial auctions. In *EC '08: Proceedings of the 9th ACM conference on Electronic commerce*, pages 70–77, New York, NY, USA, 2008. ACM.

17 Noam Nisam, Tim Roughgarden, Éva Tardos, and Vijay V. Vazirani. *Algorithmic Game Theory*. Cambridge University Press, September 2007.

18 Seth Pettie and Vijaya Ramachandran. An optimal minimum spanning tree algorithm. *J. ACM*, 49(1):16–34, 2002.

19 Ran Raz and Shmuel Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability pcp characterization of np. In *STOC '97: Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, pages 475–484, New York, NY, USA, 1997. ACM.

20 Zoya Svitkina and Lisa Fleischer. Submodular approximation: Sampling-based algorithms and lower bounds. In *FOCS '08: Proceedings of the 2008 49th Annual IEEE Symposium on Foundations of Computer Science*, pages 697–706, Washington, DC, USA, 2008. IEEE Computer Society.

Quasi-Random PCP and Hardness of 2-Catalog Segmentation

Rishi Saket*

Carnegie Mellon University
rsaket@cs.cmu.edu

Abstract

We study the problem of 2-Catalog Segmentation which is one of the several variants of segmentation problems, introduced by Kleinberg *et al.* [11], that naturally arise in data mining applications. Formally, given a bipartite graph $G = (U, V, E)$ and parameter r, the goal is to output two subsets $V_1, V_2 \subseteq V$, each of size r, to maximize,

$$\sum_{u \in U} \max\{|E(u, V_1)|, |E(u, V_2)|\},$$

where $E(u, V_i)$ is the set of edges between u and the vertices in V_i for $i = 1, 2$. There is a simple 2-approximation for this problem, and stronger approximation factors are known for the special case when $r = |V|/2$ [5, 16]. On the other hand, it is known to be NP-hard [11, 5, 12], and Feige [7] showed a constant factor hardness based on an assumption of average case hardness of random 3SAT.

In this paper we show that there is no PTAS for 2-Catalog Segmentation assuming that NP does not have subexponential time probabilistic algorithms, i.e. NP $\not\subseteq \cap_{\varepsilon>0}$ BPTIME(2^{n^ε}). In order to prove our result we strengthen the analysis of the Quasi-Random PCP of Khot [10], which we transform into an instance of 2-Catalog Segmentation. Our improved analysis of the Quasi-Random PCP proves stronger properties of the PCP which might be useful in other applications.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.447

1 Introduction

In Computer Science many important problems are known to be NP-hard, i.e. a polynomial algorithm for any of these problems will imply P = NP. Many of these problems are in essence optimization questions, for example the Max-3SAT problem of satisfying the maximum number of clauses of a 3SAT instance. It follows from the NP-hardness of SAT that it is NP-hard to compute the optimum of Max-3SAT as well. This, however, motivates the study of efficient approximation algorithms for optimization problems. An algorithm (for a minimization problem) is said to have an approximation factor of $C > 1$ if it computes a solution which is at most factor C away from the optimum; and the definition is analogous for a maximization problem so that the approximation factor C is always greater than 1. An optimization problem is said to admit a Polynomial Time Approximation Scheme (PTAS) if it has a $1 + \varepsilon$ approximation algorithm for every constant $\varepsilon > 0$, which runs in time polynomial in the size of the problem. Several important problems have been found to admit a PTAS, such as the classic Knapsack [15] and EuclideanTSP [2] problems.

For a long time it was an important open question as to whether Max-3SAT has a PTAS, until the well known Probabilistically Checkable Proof (PCP) Theorem [4, 3] proved

* Supported in part by Venkatesan Guruswami's Packard Fellowship

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 447–458

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a constant factor hardness for Max-3SAT . Equivalently, the PCP Theorem shows the existence of an efficient probabilistic verifier for NP : i.e. for any language L in NP, there is a verifier which can efficiently decide whether $x \in L$, given a proof whose size is polynomial in $|x|$. The verifier reads only a constant number of bits from the proof, uses only logarithmic (in $|x|$) randomness, and for every correct statement there is a proof such that the verifier always accepts it, while every proof for an incorrect statement is rejected with high probability. The PCP Theorem, along with tools such as Fourier Analysis and Parallel Repetition [13], has led to several important inapproximability results (many of them optimal) such as [9], [6], [8].

However, till some time ago, there remained some important problems such as Graph Min-Bisection, Dense k-Subgraph and Bipartite Clique for which no hardness of approximation was known. Feige [7] showed that these problems do not have a PTAS under the assumption that random 3SAT instances are hard on average. Subsequently, in a breakthrough work Khot [10] constructed the so called Quasi-Random PCP and used it to rule out PTAS for the above mentioned problems under the standard assumption that NP does not have subexponential time (randomized) algorithms, i.e. NP $\not\subseteq \cap_{\varepsilon>0}$ BPTIME(2^{n^ε}). More recently, the Quasi-Random PCP was used by [1] to rule out PTAS for the well known Sparsest Cut problem.

In this paper we study the notorious question (as far as inapproximability results go) of 2-CatalogSegmentation. In this problem, one is given a set of items and a set of customers, where every customer is interested in a personal subset of items. Given a parameter r, the goal is to construct two *catalogs* of r items each, and send exactly one of the two catalogs to each customer. The payoff from any customer is the number of items on the catalog sent to him that he is interested in, and the goal is to maximize the total payoff for all the customers. 2-CatalogSegmentation is one of the several variants of segmentation problems first studied by Kleinberg, Papadimitriou and Raghavan [11]. Such problems arise naturally in data mining for devising marketing strategies or production plans. It has also been used for modeling certain coding theory problems [12].

The 2-CatalogSegmentation problem is known to be NP-hard [11], [5] and [12], while there is a simple 2-approximation for it. Dodis, Guruswami and Khanna [5] studied the special case when $r = n/2$, where n is the total number of items and gave a 1.76-approximation algorithm for this special case, which was subsequently improved to 1.56 in [16]. Feige [7] showed a constant factor hardness for 2-CatalogSegmentation under an assumption about average case hardness of random 3SAT. However, under standard complexity assumptions, no hardness of approximation was known till now for the 2-CatalogSegmentation problem.

In this paper we prove a hardness of approximation result for 2-CatalogSegmentation, which is stated informally below.

▶ **Theorem.** *There is no PTAS for the* 2-CatalogSegmentation *problem unless NP has subexponential time randomized algorithms, i.e. unless NP* $\subseteq \cap_{\varepsilon>0}$ *BPTIME(*2^{n^ε}*).*

In order to prove our result we strengthen the analysis of the Quasi-Random PCP of Khot [10], which is then reduced to an instance of 2-CatalogSegmentation. Our improved analysis of the Quasi-Random PCP proves stronger properties of the PCP which might be useful in other applications.

In the next section we start with some preliminary definitions and statement of our results. In Section 3 we shall prove the inapproximability of 2-CatalogSegmentation based on the properties of the Quasi-Random PCP obtained by our strengthened analysis. The subsequent sections are devoted to proving the desired properties of the Quasi-Random PCP.

2 Preliminaries

We start with the formal definition of the 2-CatalogSegmentation problem.

▶ **Definition 1.** 2-CatalogSegmentation: Given a bipartite graph $G = (U, V, E)$, and a parameter r, the goal is to output two sets V_1 and V_2 such that $V_1, V_2 \subseteq V$ and $|V_1|, |V_2| = r$ to maximize the following quantity.

$$\sum_{u \in U} \max \{|E(u, V_1)|, |E(u, V_2)|\}.$$

where $E(u, V_i)$ is the set of edges incident on u with the other end in V_i, for $i = 0, 1$.

The vertices in U represent the customers and the ones in V represent the items, and an edge (u, v) signifies that the customer u is interested in item v. One is required to construct two catalogs V_1 and V_2 of r items each and send each customer one of the two catalogs. The objective is to maximize the sum, over all customers, of the number of items each customer receives in his catalog that he is interested in. Feige [7] proved a conditional inapproximability result for 2-CatalogSegmentation under a hypothesis about the average case hardness of 3SAT, both of which are stated below.

▶ **Hypothesis 2.** *(Random* 3SAT *Hypothesis) For every fixed $\varepsilon > 0$ and for Δ a sufficiently large constant independent of n, there is no polynomial time algorithm that on a random* 3CNF *formula with n variables and $m = \Delta n$ clauses, outputs YES if the formula is satisfiable, and NO at least half the time if the formula is unsatisfiable.*

▶ **Theorem 3.** *(Feige [7]) Assuming Hypothesis 2, there is no polynomial time algorithm to approximate* 2-CatalogSegmentation *within a factor of $1 + \varepsilon$ for some $\varepsilon > 0$.*

Feige [7] also proved inapproximability results for other problems such as Graph Min-Bisection, Dense k-subgraph and Bipartite Clique based on Hypothesis 2. As mentioned in the previous section, Khot [10] subsequently constructed the Quasi-Random PCP, which was used to rule out PTAS for Graph Min Bisection, Dense k-subgraph and Bipartite Clique, under the standard assumption that NP has no subexponential time algorithms. Before proceeding, we recall the formal statement of the Quasi-Random PCP.

▶ **Theorem 4.** *(Khot's Quasi-Random PCP [10]) For every $\varepsilon > 0$, there exists an integer $d = O(1/\varepsilon \log(1/\varepsilon))$ such that the following holds : there is a PCP verifier for a* SAT *instance of size n satisfying:*

1. *The proof for the verifier is of size $2^{O(n^\varepsilon)}$.*
2. *The verifier reads $4d$ bits from the proof. Let Q be the $4d$ bits queried by the verifier in a random test.*
3. *Every query bit is uniformly distributed over the proof, though different query bits within Q are correlated.*
4. *(YES Case) Suppose that the* SAT *instance is satisfiable. Then there exists a correct proof Π^*, such that if Π^*_0 be the set of 0-bits in the proof Π^*, then,*

$$\Pr_Q [Q \subseteq \Pi^*_0] \geq D \frac{1}{2^{4d-1}},$$

where $D = \left(1 - O\left(\frac{1}{d^2}\right)\right)$ and the probability is taken over a random test of the verifier.

5. *(NO Case) Suppose that the* SAT *instance is unsatisfiable, and let* Π' *be any set of half the bits in the proof. Then,*

$$\left|\Pr_Q[Q \subseteq \Pi'] - \frac{1}{2^{4d}}\right| \leq \frac{1}{2^{40d}}.$$

As one can see, in the NO case the PCP exhibits a quasi-randomness property, in the sense that the probability is close to what is expected if each query bit were chosen uniformly at random in the proof. However, the above statement does not seem strong enough to prove an inapproximability for the 2-CatalogSegmentation problem. In our results we prove a strengthened statement for the Quasi-Random PCP and apply it to prove the desired result for 2-CatalogSegmentation. In the next few paragraphs we formally state the results of this paper.

2.1 Our Results

For the purpose of convenience, we let the query Q of the PCP verifier be a tuple of $4d$ bits in the proof, i.e. $Q = (q_1, q_2, \ldots, q_{4d})$. The verifier queries the bits q_i $(1 \leq i \leq 4d)$ as part of the query Q. For a given proof Π, let $val(\Pi, q)$ denote the 0 or 1 value of the proof Π at the bit q. We prove the following strengthened theorem regarding the Quasi-Random PCP.

▶ **Theorem 5.** *For every* $\varepsilon > 0$*, there exists an integer* $d = O(1/\varepsilon \log(1/\varepsilon))$ *such that the following holds : there is a PCP verifier for a* SAT *instance of size* n *satisfying the following properties.*

1. *The proof for the verifier is of size* $2^{O(n^\varepsilon)}$.
2. *The verifier reads* $4d$ *bits from the proof. Let* $Q = (q_1, \ldots q_{4d})$ *be the tuple of* $4d$ *bits queried by the verifier.*
3. *Every query bit* q_i *is uniformly distributed over the proof, though different query bits within* Q *are correlated.*
4. *(YES Case) Suppose that the* SAT *instance is satisfiable. Then there exists a correct proof* Π^**, which is* 1 *on exactly half the fraction of the bits and satisfies the following property. Fix any* $4d$ *boolean values* $r_1, r_2, \ldots, r_{4d} \in \{0,1\}$*, such that* $\sum_{i=1}^{4d} r_i = 0 \pmod 2$*. Then,*

$$\Pr_Q\left[\bigwedge_{i=1}^{4d}(val(\Pi^*, q_i) = r_i)\right] \geq D\frac{1}{2^{4d-1}},$$

   where $D = \left(1 - O\left(\frac{1}{d^2}\right)\right)$ *is independent of* $r_1, \ldots, r_{4d}$*, and the probability is taken over a random test of the verifier.*
5. *(NO Case) Suppose that the* SAT *instance is unsatisfiable, and let* Π *be any proof that is* 1 *on exactly half fraction of the bits. Fix any* $4d$ *boolean values* $s_1, s_2, \ldots, s_{4d} \in \{0,1\}$*. Then,*

$$\left|\Pr_Q\left[\bigwedge_{i=1}^{4d}(val(\Pi, q_i) = s_i)\right] - \frac{1}{2^{4d}}\right| \leq \frac{1}{2^{40d}}.$$

Note that in the above statement, we prove a stronger property of the Quasi-Random PCP in the YES case, which is that the distribution of the $4d$ bit string read in the query Q is close to that of a uniform distribution over all $4d$ bit strings with an even number of 1s. This implies the property in the YES case proved in Theorem 4. In some sense we prove a partial quasi-randomness property even in the YES case, except that it is with respect to the uniform distribution over all $4d$ bit strings with even number of 1s. The property that we prove in the NO case is also a similar generalization of the corresponding property in

Theorem 4. Essentially, the distribution of the $4d$ bit string read by the query Q is close to what is obtained by picking $4d$ random bits from the proof given in the NO case.

Using the above strengthened statement of the Quasi-Random PCP we prove the following inapproximability of the 2-CatalogSegmentation problem.

▶ **Theorem 6.** *Let $\varepsilon > 0$ be an arbitrarily small constant. Assume that* SAT *has no algorithm in BPTIME(2^{n^ε}). Then there is no polynomial time algorithm for* 2-CatalogSegmentation *that achieves an approximation of $1 + \Omega(\frac{1}{d})$, where $d = O(1/\varepsilon \log(1/\varepsilon))$. In particular, the* 2-CatalogSegmentation *problem does not admit a PTAS unless NP $\subseteq \cap_{\varepsilon>0}$ BPTIME(2^{n^ε}).*

A sketch of the proof of Theorem 5 is given in Sections 4, 5 and 6. It requires describing, at least to some extent, the construction of the Quasi-Random PCP of [10]. We start with the description of HomAlgCSP problem in Section 4. This is the starting point for constructing an Outer Verifier in Section 5 and the final Inner Verifier in Section 5.2. The construction of these verifiers is same as in [10] except for some convenient notational changes and appropriate selection of parameters. Section 6 gives a brief sketch of analysis of the PCP, a key new ingredient in which is Lemma 11 that is used along with the techniques of [10] to prove the strengthened property in the YES case.

In the next section we prove Theorem 6 assuming Theorem 5. We reduce from the Quasi-Random PCP to an instance of 2-CatalogSegmentation. The reduction is similar to the one used to prove Theorem 3 in [7].

3 Reduction to 2-CatalogSegmentation and proof of Theorem 6

In this section we describe the reduction to 2-CatalogSegmentation from the Quasi-Random PCP given by Theorem 5. In the following construction, U and V will be the sets of customers and items respectively. There is an edge between a customer and an item if the customer is interested in that item. The reduction is as follows.

1. Let the set of customers U be the set of all the bits in the proof of the PCP.
2. For every tuple of $4d$ bits Q queried by the PCP verifier we have an item. We replicate every item (tuple of $4d$ bits) proportional to the probability it is queried by the verifier. Let V be the set of all items.
3. A bit in U is connected to all the tuples Q in V that contain it.
4. Set $r = D|V|\frac{1}{2^{4d-1}}$ to be the catalog size. Here $D = \left(1 - O\left(\frac{1}{d^2}\right)\right)$ as in the YES case of Theorem 5.

The analysis is as follows.

YES Case. Let Π^* be the correct proof to the PCP verifier given by Theorem 5. Construct two catalogs $V_1, V_2 \subseteq V$ where,

$$V_1 \subseteq \{Q \mid \text{all bits of } Q \text{ are set to 0 in } \Pi^*\}, \tag{1}$$

$$V_2 \subseteq \{Q \mid \text{all bits of } Q \text{ are set to 1 in } \Pi^*\} \tag{2}$$

Setting $r_i = 0$ for $i = 1, \ldots, 4d$ in the property of the YES case in Theorem 5, we can ensure that,

$$|V_1| = D|V|\frac{1}{2^{4d-1}}.$$

Similarly, by setting $r_i = 1$ for $i = 1, \ldots, 4d$ we can ensure that,

$$|V_2| = D|V|\frac{1}{2^{4d-1}}.$$

We note that both V_1 and V_2 are disjoint subsets of V.

Now send catalog V_1 to the customers corresponding to the bits set to 0 in Π^* and V_2 to the complement, i.e. customers corresponding to the bits set to 1 in Π^*. Clearly, each item in V_1 and V_2 reaches $4d$ customers interested in it. Therefore the value of the solution is,

$$\begin{aligned} & 4d|V|2D\left(\frac{1}{2^{4d-1}}\right) \\ = \; & 8d|V|\left(1-O\left(\frac{1}{d^2}\right)\right)\left(\frac{1}{2^{4d-1}}\right) \end{aligned} \tag{3}$$

NO Case. For convenience we allow the two catalogs to be of size $|V|\frac{1}{2^{4d-1}}$, as this can only increase the payoff. In the NO Case, one of the catalogs, call it V' reaches at most half of the customers. Clearly, the payoff obtained by this catalog only increases if we enlarge the set of customers to which V' is sent, without changing the other catalog and the set of customers to which it is sent. Therefore, we may assume that V' is sent to exactly half of the customers. Let the proof Π be constructed by setting the bits corresponding to these customers to be 1 and the rest to 0. From the NO case of Theorem 5, by setting $s_i = 1$ for $i = 1, \ldots, 4d$, we obtain that at most $\frac{1}{2^{4d}} + \frac{1}{2^{40d}}$ fraction of all the tuples Q queried have all the $4d$ bits set to 1. Therefore, there are at most $\left(\frac{1}{2^{4d}} + \frac{1}{2^{40d}}\right)|V|$ items in V' that reach $4d$ customers interested in them. Therefore, at least $\left(\frac{1}{2^{4d}} - \frac{1}{2^{40d}}\right)|V|$ items in V' reach at most $4d-1$ customers interested in them. The other catalog has a payoff of at most $4d|V|\frac{1}{2^{4d-1}}$. Hence, the value of any solution in the NO case is at most,

$$\begin{aligned} & |V|\left(4d\left(\frac{1}{2^{4d-1}}\right) + 4d\left(\frac{1}{2^{4d}} + \frac{1}{2^{40d}}\right) + (4d-1)\left(\frac{1}{2^{4d}} - \frac{1}{2^{40d}}\right)\right) \\ \leq \; & |V|\left(4d\left(\frac{1}{2^{4d-1}}\right) + 4d\left(\frac{3}{2}\cdot\frac{1}{2^{4d}}\right) + (4d-1)\left(\frac{1}{2}\cdot\frac{1}{2^{4d}}\right)\right) \\ = \; & 8d|V|\left(1-\frac{1}{32d}\right)\left(\frac{1}{2^{4d-1}}\right). \end{aligned} \tag{4}$$

Hardness Factor. From Equations (3) and (4) we obtain that the ratio of the value of the solution in the YES case to the best solution in the NO case is at least,

$$\frac{\left(1-O\left(\frac{1}{d^2}\right)\right)}{\left(1-\frac{1}{32d}\right)} = 1+\Omega\left(\frac{1}{d}\right) \tag{5}$$

Therefore, if 2-CatalogSegmentation is approximable within factor $1+\Omega\left(\frac{1}{d}\right)$, then NP $\subseteq$ BPTIME(2^{n^ε}), where $d = O(1/\varepsilon \log 1/\varepsilon)$. This rules out PTAS for 2-CatalogSegmentation unless NP $\subseteq \cap_{\varepsilon>0}$BPTIME$(2^{n^\varepsilon})$. This proves Theorem 6.

4 Homogeneous Algebraic CSP

We define the HomAlgCSP problem which is the starting point of the reduction in this paper. This definition is a (slightly modified) restatement of Definition 3.1 of [10].

▶ **Definition 7.** Given parameters k, d, m and a field $\mathbb{F}$, let HomAlgCSP instance $\mathcal{A}(k, d, m, \mathbb{F}, \mathcal{C})$ be the following problem (think of k as a fixed integer like 21, and d as a large constant integer) :

1. $\mathcal{C}$ is a system of constraints on functions $f : \mathbb{F}^m \mapsto \mathbb{F}$ where every constraint is on values of f on k different points and is given by a conjunction of homogeneous linear constraints on those k values. A typical constraint $C \in \mathcal{C}$ looks like

$$\sum_{i=1}^{k} \gamma_{ij} f(\overline{p}_i) = 0 \quad \text{for } j = 1, 2, \ldots \qquad \text{where } \overline{p}_i \in \mathbb{F}^m \text{ and } \gamma_{ij} \in \mathbb{F}.$$

We denote a constraint C by the set of points $\{\overline{p}_i\}_{i=1}^k$, while the γ_{ij}'s will be implicit.

2. $\mathcal{C}$ has $|\mathbb{F}|^{O(m)}$ constraints.

The goal is to find a m-variate polynomial f, not identically zero, so as to maximize the fraction of constraints satisfied. Let $OPT(\mathcal{A})$ denote the maximum fraction of constraints satisfied by any such polynomial of degree at most d.

5 Construction of the PCP

This section describes construction given in [10] of a PCP Outer Verifier and Inner Verifiers for a HomAlgCSP instance $\mathcal{A}(k = 21, d^*, m, \mathbb{F}, \mathcal{C})$ based on a variant of the Low-Degree test of Rubinfeld and Sudan [14].

The Outer Verifier is given the polynomial f as a table of values at each point in $\mathbb{F}^m$. It picks a constraint in $\mathcal{C}$ uniformly at random and checks whether it is satisfied. Before the description we need the definition of a curve.

▶ **Definition 8.** A *curve* L in $\mathbb{F}^m$ is a function $L : \mathbb{F} \mapsto \mathbb{F}^m$, where $L(t) = (a_1(t), \ldots, a_m(t))$. It is a degree d curve if each of the coordinate functions a_i $(1 \leq i \leq m)$ is degree d (univariate) polynomial.

A *line* is a curve of degree 1.

Let $C(\{\overline{p}_i\}_{i=1}^k) \in \mathcal{C}$, denote the constraint that the Verifier chooses at random to check. Let $t_1, t_2, \ldots, t_{k+3}$ be distinct field elements in $\mathbb{F}$ which we fix for the rest of the paper. For $\overline{a}, \overline{b}, \overline{c} \in \mathbb{F}^m$, let $L = L_{\overline{a},\overline{b},\overline{c}}$ be the unique degree $k+2$ curve that passes through the points $\{\overline{p}_i\}_{i=1}^k, \overline{a}, \overline{b}, \overline{c}$. More precisely,

$$L(t_i) = \overline{p}_i, \quad 1 \leq i \leq k, \quad L(t_{k+1}) = \overline{a}, \quad L(t_{k+2}) = \overline{b}, \quad L(t_{k+3}) = \overline{c}.$$

In brief the strategy of the Outer Verifier is as follows. Suppose f is a degree d^* multivariate polynomial over the vector space $\mathbb{F}^m$. Clearly, its restriction to the curve $L(t) = L_{\overline{a},\overline{b},\overline{c}}(t)$ is a degree $d - 1 := (k+2)d^*$ univariate polynomial in t. This polynomial, denoted by $f|_L$, can be interpolated from any d values of f on the curve. This is precisely what the verifier does: it picks $d+1$ points on the curve L, interpolates $f|_L$ from the first d points and verifies that the value of f and $f|_L$ is the same on the last point. In addition, it checks that the values of $f|_L$ at the points $\{\overline{p}_i\}_{i=1}^k$ satisfies the constraint C. Note that the values $t_1, \ldots, t_{k+3}$ on which L depends, are fixed. This is combined with the line-point Low Degree Test. Given a line ℓ, the restriction of f, denoted by $f|_\ell$ is a degree d^* univariate polynomial, but we allow it degree up to $d-2$, and interpolate it using the values of f at $d-1$ random points on ℓ.

We next give the detailed description of the *Modified Outer Verifier*, which for technical reasons, reads more values from the proof and makes additional tests, while building upon the Outer Verifier. Also, it abstracts out the tasks of interpolation into multiplication by an invertible matrix, and checking the homogeneous constraints of the Outer Verifier into checking orthogonality with a certain subspace.

5.1 Modified Outer Verifier

We first observe that $\mathbb{F}$ is an extension of $\mathbb{F}[2]$. Therefore, we can represent the elements of $\mathbb{F}$ as bit strings of a length $l = \log|\mathbb{F}|$. Moreover, the representation can be chosen such that addition over $\mathbb{F}$ and multiplication by a constant in $\mathbb{F}$ are homogeneous linear operations on these bit strings. The Modified Outer Verifier is given a table of values $f(\overline{v})$ (in the form of l bit strings) for every point $\overline{v} \in \mathbb{F}^m$ and it executes the following steps:

Steps of the Modified Outer Verifier

1. Pick a constraint $C = \{\overline{p}_i\}_{i=1}^{k} \in \mathcal{C}$ at random.
2. Pick a random line ℓ (in $\mathbb{F}^m$) and pick random points $\overline{v}_1, \ldots, \overline{v}_{d-1}, \overline{v}_d$ on the line.
3. Pick $t \in \mathbb{F} \setminus \{t_1, \ldots, t_{k+3}\}$ at random, points $\overline{a}, \overline{b}$ at random from $\mathbb{F}^m$ and let L be the unique degree $k+2$ curve $L = L_{\overline{a},\overline{b},\overline{c}}$ such that, $L(t_i) = \overline{p}_i, \quad 1 \le i \le k$, $L(t_{k+1}) = \overline{a}$, $L(t_{k+2}) = \overline{b}$, and $L(t) = \overline{v}_d$ so that $\overline{c}$ is automatically defined to $L(t_{k+3})$.
4. Pick random points $\overline{v}_{d+1}, \ldots, \overline{v}_{2d}$ on the curve L.
5. Pick additional random points $\overline{u}_1 \ldots \overline{u}_d$ on the line ℓ and $\overline{u}_{d+1}, \ldots, \overline{u}_{2d}$ from the curve L. (We assume that all the points chosen on the line ℓ and curve L are distinct, which happens w.h.p)
6. Let $T_{2ld \times 2ld}$ be an appropriate invertible matrix over $\mathbb{F}[2]$ and H be an appropriate subspace of $\mathbb{F}[2]^{2ld}$. Both depend only on the choice of the points $\{\overline{v}_i\}_{i=1}^{2d}$ and $\{\overline{u}_j\}_{j=1}^{2d}$. Remark 1 explains how T and H are chosen.
7. Read the values of the function f from the table at the points $\overline{v}_1, \ldots, \overline{v}_{2d}$ and $\overline{u}_1, \ldots, \overline{u}_{2d}$. Since all the elements of the field are represented by bit strings, let
$$x = f(\overline{v}_1) \circ f(\overline{v}_2) \circ \cdots \circ f(\overline{v}_{2d}) \tag{6}$$
$$y = f(\overline{u}_1) \circ f(\overline{u}_2) \circ \cdots \circ f(\overline{u}_{2d}) \tag{7}$$
where $\circ$ represents concatenation of strings.
8. Accept iff,
$$x \neq 0,\ x = Ty \qquad \text{and} \qquad h \cdot x = 0 \quad \forall\ h \in H \ \ (\text{i.e. } x \perp H). \tag{8}$$

▶ **Remark 1.** *The subspace H is chosen such that the constraint $h \cdot x = 0\ \forall\ h \in H$ ensures that the values at the field elements $\{t_i\}_{i=1}^{k}$ of the degree $d-1$ univariate polynomial interpolated from $f(\overline{v}_{d+1}) \ldots f(\overline{v}_{2d})$, (which are supposed to be the values of f at $\{\overline{p}_i\}_{i=1}^{k}$) satisfy the homogeneous linear constraints of C. In addition, H is chosen such that the polynomial interpolated from the values $f(\overline{v}_1) \ldots f(\overline{v}_{d-1})$ agrees with the degree $d-1$ polynomial interpolated from $f(\overline{v}_{d+1}) \ldots f(\overline{v}_{2d})$ at the point $\overline{v}_d$, where both evaluate to $f(\overline{v}_d)$.*

The invertible matrix T is chosen such that the constraint $x = Ty$ ensures the following conditions are satisfied:

1. *The degree $d-1$ polynomial interpolated from the values $f(\overline{v}_1) \ldots f(\overline{v}_d)$ is the same as the polynomial interpolated from the values $f(\overline{u}_1) \ldots f(\overline{u}_d)$. (This polynomial will actually be of degree $d-2$ due to the constraint enforced by the subspace H).*
2. *The degree $d-1$ polynomial interpolated from $f(\overline{v}_{d+1}) \ldots f(\overline{v}_{2d})$ is the same as the polynomial interpolated from the values $f(\overline{u}_{d+1}) \ldots f(\overline{u}_{2d})$.*

The condition $x \neq 0$ essentially ensures that f is not a zero polynomial.

5.2 Inner Verifier

We now construct the Inner Verifier which is essentially identical to the one constructed in [10], except for some notational complications that we need to introduce. It expects, for

every point $\overline{v} \in \mathbb{F}^m$, the Hadamard Code of the string $f(\overline{v}) \in \{0,1\}^l$ (refer to Appendix A.3 of [10] for an overview). The following are the steps executed by the verifier.

Steps of the Inner Verfier

1. Pick a constraint $C \in \mathcal{C}$ and the points $\overline{v}_1, \ldots, \overline{v}_{2d}$ and $\overline{u}_1, \ldots, \overline{u}_{2d}$ as in steps $1-5$ of the Modified Outer Verifier.
2. Let $T_{2ld \times 2ld}$ and H be the matrix and subspace respectively chosen as in step 7 of the Modified Outer Verifier.
3. Pick a random string $z \in \mathbb{F}^{2ld}$ and a random $h \in H$. Write, $z = z_1 \circ z_2 \circ \cdots \circ z_{2d}$, $h = h_1 \circ h_2 \circ \cdots \circ h_{2d}$, and $zT = w_1 \circ w_2 \circ \cdots \circ w_{2d}$.
4. Let $A_1, \ldots, A_{2d}$ and $B_1, \ldots, B_{2d}$ be the (supposed) Hadamard Codes of $f(\overline{v}_1), \ldots, f(\overline{v}_{2d})$ and
   $f(\overline{u}_1), \ldots, f(\overline{u}_{2d})$ respectively, given by the proof Π.
5. Let Q be defined as the tuple of $4d$ 'positions' queried by the Inner Verifier. It is formally set as: $Q = (q_1, \ldots, q_{2d}, q_{2d+1} \ldots, q_{4d})$, where q_i is the bit read at the position $z_i \oplus h_i$ of the Hadamard Code A_i for $1 \leq i \leq 2d$. Similarly, q_{j+2d} is the bit read at position w_j of the Hadamard Code B_j for $1 \leq j \leq 2d$.
6. Let $val(q_i, \Pi) \in \{0,1\}^{4d}$ be the value of the i^{th} bit in the tuple Q given by the proof Π. From the construction of Π and Q we have: $val(q_i, \Pi) = A_i(z_i \oplus h_i), \quad 1 \leq i \leq 2d$, and $val(q_{j+2d}, \Pi) = B_j(w_j), \quad 1 \leq j \leq 2d$.
6. Accept iff $\oplus_{i=1}^{4d} val(q_i, \Pi) = 0$.

For our eventual application, we are in fact not interested in the acceptance probabilities of the Inner Verifier in the YES and NO cases. Instead, we wish to study the distribution of the number of 1s and 0s in the tuple of $4d$ bits Q.

6 Sketch of Analysis

We begin this sketch by first stating the two key lemmas regarding the behaviour of the Inner Verifier depending on the instance $\mathcal{A}$ of HomAlgCSP.

The first lemma states that if the instance $\mathcal{A}$ of HomAlgCSP has a very good optimum, then there is a proof to the Inner Verifier such that the distribution of the $4d$ bits of Q, read by the verifier from the proof, is close to the uniform distribution over $4d$ bit strings with even number of 1s.

▶ **Lemma 9.** *Let $r_1, \ldots, r_{4d} \in \{0,1\}$ be any fixed boolean values such that $\sum_{i=1}^{4d} r_i = 0 \pmod 2$. Suppose the $\mathcal{A}$ is an instance of* HomAlgCSP *with optimum $OPT(\mathcal{A})$, given by the polynomial f. Let Π^* be the proof (for the Inner Verifier) constructed taking the Hadamard Code for every value of f. Let $Q = (q_1, \ldots, q_{4d})$ be the (random) tuple of length $4d$ bits queried, as described in the steps of the Inner Verifier. Similarly, let $val(q_i, \Pi^*)$ be the value in the proof Π^* at the i^{th} bit in Q. Then,*

$$\Pr_Q \left[\bigwedge_{i=1}^{4d} (val(q_i, \Pi^*) = r_i) \right] \geq OPT(\mathcal{A}) \cdot \frac{1}{2^{4d-1}}, \tag{9}$$

where the probability is taken over the random test of the Inner Verifier.

Note that in the above lemma, the proof Π^* is balanced i.e. it is 1 on half fraction of the bits. This is because Hadamard Codes of non-zero values are balanced and since f is not identically zero and has degree at most d^*, it is non-zero on all except a negligibly small fraction of points in $\mathbb{F}^m$.

The next lemma states that if there is a proof to the Inner Verifier such that if the distribution of the $4d$ bits of Q (read by the verifier from the proof) deviates significantly from the uniform distribution over $4d$ bit strings from the proof, there is a constant degree polynomial that satisfies a significant fraction of constraints of $\mathcal{A}$.

▶ **Lemma 10.** *Let $\mathcal{A}$ be an instance of* HomAlgCSP *and suppose Π is a proof to the Inner Verifier for $\mathcal{A}$, with the property that Π is 1 on exactly half fraction of the total bits. As before, let $Q = (q_1, \ldots, q_{4d})$ be the tuple of $4d$ bits queried by the Inner Verifier, and $val(q_i, \Pi)$ be the value in the proof Π at the i^{th} bit of the tuple Q. Let $s_1, \ldots, s_{4d} \in \{0,1\}$ be any $4d$ boolean values. If,*

$$\left| \Pr_{Q} \left[\bigwedge_{i=1}^{4d} (val(q_i, \Pi) = r_i) \right] - \frac{1}{2^{4d}} \right| \geq \delta \geq 0 \tag{10}$$

then there is a polynomial of degree at most $50d^*$, not identically zero, which satisfies $\delta^{C'}$ fraction of the constraints of $\mathcal{A}$, where C' is an absolute constant.

The proof of Lemma 10 follows from Theorem 7.6 of [10] which bounds the acceptance probability of the Modified Outer Verifier. The proof of Theorem 5 follows from the above two lemmas combined with Theorem 3.4 of [10] which proves the inapproximability of HomAlgCSP. The various parameters need to be chosen appropriately and the analysis follows the same scheme as given in Section 10 of [10].

Our main contribution is to strengthen the analysis in the YES case of Theorem 5, which is done by proving a more general Lemma 9 as compared to Lemma 10.2 of [10]. The main ingredient is the following lemma which we state and prove below. Before we do so, let us recall some notation.

We shall consider a test of the Modified Outer Verifier. Let T be the $2ld \times 2ld$ invertible matrix over $\mathbb{F}[2]$ and H be the appropriate subspace of $\mathbb{F}[2]^{2ld}$ constructed by the Modified Outer Verifier depending on the (randomized) choice of the constraint C, line ℓ, curve L and the points $\overline{v}_1, \ldots, \overline{v}_{2d}, \overline{u}_1, \ldots, \overline{u}_{2d}$, as explained in Remark 1. Note that the values queried by the Modified Outer Verifier are represented by l-bit strings over $\mathbb{F}[2]$.

▶ **Lemma 11.** *Let C be a constraint in $\mathcal{C}$ that is satisfied by the polynomial f (of degree at most d^*) given by Lemma 9. Let $\alpha, \beta \in \mathbb{F}[2]^{2ld}$ such that $\alpha := \alpha_1 \circ \cdots \circ \alpha_{2d}$ and $\beta := \beta_1 \circ \cdots \circ \beta_{2d}$ where $\alpha_i, \beta_j \in \mathbb{F}[2]^l$ for $1 \leq i, j \leq 2d$. Also, let property* `P1` *for α and β be defined as follows.*

`P1`: *Each α_i is either 0 or $f(\overline{v}_i)$ for $1 \leq i \leq 2d$, and each β_j is either 0 or $f(\overline{u}_j)$ for $1 \leq j \leq 2d$.*

Then with probability $1 - O(d^2/|\mathbb{F}|)$ over the choice of the line ℓ, curve L, and the points $\{\overline{v}_i\}_{i=1}^{2d}$ and $\{\overline{u}_j\}_{j=1}^{2d}$, the following holds:
The only two solutions to $\alpha \perp H$, $\beta = T^{-1}\alpha$ satisfying property `P1` *are,*

$$\alpha_i = \beta_j = 0 \quad \forall 1 \leq i, j \leq 2d, \tag{11}$$

and,

$$\alpha_i = f(\overline{v}_i), \quad \beta_j = f(\overline{u}_j) \;\; \forall 1 \leq i, j \leq 2d. \tag{12}$$

Proof. Firstly, we have that with probability at least $1 - O(d^2/|\mathbb{F}|)$, none of the values $\{f(\overline{v}_i)\}_{i=1}^{2d}$ and $\{f(\overline{u}_j)\}_{j=1}^{2d}$ are 0. This is because each of the $4d$ points are uniformly distributed over $\mathbb{F}^m$ and since f is not identically zero and of degree at most $d^* \leq d$, by

Schwartz-Zippel Lemma the probability that any of the $4d$ points is a root of f is at most $4d \cdot O(d/|\mathbb{F}|) = O(d^2/|\mathbb{F}|)$. Since this probability is negligible, we can assume for the rest of the argument that none of the values $\{f(\overline{v}_i)\}_{i=1}^{2d}$ and $\{f(\overline{u}_j)\}_{j=1}^{2d}$ are 0.

Next, from the construction of the matrix T (refer to Remark 1) we have that $\beta = T^{-1}\alpha$ implies the following two properties.

P2: The polynomial interpolated by the values α_i at point $\overline{v}_i$ for $1 \leq i \leq d$ is identical to the one interpolated by the values β_j at point $\overline{u}_j$ for $1 \leq j \leq d$.

P3: The polynomial interpolated by the values α_i at point $\overline{v}_i$ for $d+1 \leq i \leq 2d$ is identical to the one interpolated by the values β_j at point $\overline{u}_j$ for $d+1 \leq j \leq 2d$.

Also, from the construction of the subspace H (refer to Remark 1) we have that $\alpha \perp H$ implies the following property.

P4: The polynomial interpolated from the values α_i at points $\overline{v}_i$ $(1 \leq i \leq d-1)$ agrees with the polynomial interpolated from the values α_j at points $\overline{v}_j$ $(d+1 \leq j \leq 2d)$ at the point $\overline{v}_d$ where both evaluate to α_d.

Clearly the solution $\alpha_i = \beta_j = 0$ for all $1 \leq i, j \leq 2d$ is a valid solution to $\alpha \perp H, \alpha = T^{-1}\beta$ and satisfying property P1. Also, since f is a degree d^* polynomial that satisfies the constraint C, the solution $\alpha_i = f(\overline{v}_i)$ and $\beta_j = f(\overline{u}_j)$ for all $1 \leq i, j \leq 2d$ is a valid solution as well.

Suppose that there is another solution α, β, different from the above two, satisfying the properties P1, P2, P3 and P4. Then, at least one of the following eight cases must happen, all of which we show have a low probability of occurring.

Case 1. $\alpha_i = 0$ for $1 \leq i \leq d$ and $\alpha_j = f(\overline{v}_j)$ for $d+1 \leq j \leq 2d$. This along with property P4 implies that the univariate f' polynomial interpolated from the values $\alpha_j = f(\overline{v}_j)$ at points $\overline{v}_j$ for $d+1 \leq j \leq 2d$ evaluates to $\alpha_d = 0$ at the point $\overline{v}_d$. Clearly, since f' is unique, it has to evaluate to $f|_L(\overline{v}_d) = f(\overline{v}_d)$ at point $\overline{v}_d$. This implies that $f(\overline{v}_d) = 0$ which contradicts our earlier assumption that all the values $f(\overline{v}_i)$ and $f(\overline{u}_j)$ $(1 \leq i, j \leq 2d)$ are non zero.

Case 2. $\alpha_i = f(\overline{v}_i)$ for $1 \leq i \leq d$ and $\alpha_j = 0$ for $d+1 \leq j \leq 2d$. Again, property P4 implies that zero polynomial interpolated from the values $\alpha_j = 0$ at points $\overline{v}_j$ for $d+1 \leq j \leq 2d$, evaluates to $\alpha_d = f(\overline{v}_d)$ at point $\overline{v}_d$. This means that $f(\overline{v}_d) = 0$, which is a contradiction to our assumption.

Case 3. $\beta_i = 0$ for $1 \leq i \leq d$ and $\beta_j = f(\overline{u}_j)$ for $d+1 \leq j \leq 2d$. From properties P2 and P3, this implies that $\alpha_i = 0$ for $1 \leq i \leq d$ and $\alpha_j = f(\overline{v}_j)$ for $d+1 \leq j \leq 2d$. Thus this reduces to Case 1.

Case 4. $\beta_i = f(\overline{u}_i)$ for $1 \leq i \leq d$ and $\beta_j = 0$ for $d+1 \leq j \leq 2d$. Again, properties P2 and P3 imply that $\alpha_i = f(\overline{v}_i)$ for $1 \leq i \leq d$ and $\alpha_j = 0$ for $d+1 \leq j \leq 2d$. Thus this reduces to Case 2.

Case 5. There exist $1 \leq i', j' \leq d$ such that $\alpha_{i'} = f(\overline{v}_{i'})$ and $\beta_{j'} = 0$. Let f_1 be the univariate polynomial interpolated by the values α_i at point $\overline{v}_i$ for $1 \leq i \leq d$. This polynomial is not identically zero since $\alpha_{i'} = f(\overline{v}_{i'}) \neq 0$ (by our initial assumption). Also the degree of the polynomial is at most d. Property P2 implies that f_1 takes the value $\beta_{j'} = 0$ at the point $\overline{u}_{j'}$. However, since the points $\{\overline{u}_j\}_{j=1}^{d}$ are chosen uniformly at random on the line ℓ (refer to Section 5.1), the probability that one of them is a root of f_1 is at most $O(d^2/|\mathbb{F}|)$. Therefore, this case occurs with probability at most $O(d^2/|\mathbb{F}|)$.

Case 6. There exist $d+1 \leq i', j' \leq 2d$ such that $\alpha_{i'} = f(\overline{v}_{i'})$ and $\beta_{j'} = 0$. Let f_2 be the univariate polynomial interpolated by the values α_i at point $\overline{v}_i$ for $d+1 \leq i \leq 2d$. This polynomial is not identically zero since $\alpha_{i'} = f(\overline{v}_{i'}) \neq 0$. Also the degree of the polynomial is at most d. Property P3 implies that f_2 takes the value $\beta_{j'} = 0$

at the point $\overline{u}_{j'}$. However, since the points $\{\overline{u}_j\}_{j=d+1}^{2d}$ are chosen uniformly at random on the curve L, the probability that one of them is a root of f_2 is at most $O(d^2/|\mathbb{F}|)$. Therefore, this case also occurs with probability at most $O(d^2/|\mathbb{F}|)$.

Case 7. There exist $1 \leq i', j' \leq d$ such that $\alpha_{i'} = 0$ and $\beta_{j'} = f(\overline{u}_{j'})$. This is analogous to Case 5 and omitting the analysis we conclude that it occurs with probability at most $O(d^2/|\mathbb{F}|)$.

Case 8. There exist $d+1 \leq i', j' \leq 2d$ such that $\alpha_{i'} = 0$ and $\beta_{j'} = f(\overline{u}_{j'})$. This is analogous to Case 6 and we omit the analysis to conclude that this case occurs with probability at most $O(d^2/|\mathbb{F}|)$ as well.

Combining the above completes the proof of the lemma. ◀

References

1 C. Ambühl, M. Mastrolilli, and O. Svensson. Inapproximability results for sparsest cut, optimal linear arrangement, and precedence constrained scheduling. In *Proc.* 48^{th} *IEEE FOCS*, pages 329–337, 2007.

2 S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *J. ACM*, 45(5):753–782, 1998.

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

4 S. Arora and S. Safra. Probabilistic checking of proofs: A new characterization of NP. *J. ACM*, 45(1):70–122, 1998.

5 Y. Dodis, V. Guruswami, and S. Khanna. The 2-catalog segmentation problem. In *SODA*, pages 897–898, 1999.

6 U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

7 U. Feige. Relations between average case complexity and approximation complexity. In *Proc.* 34^{th} *ACM STOC*, pages 534–543, 2002.

8 J. Håstad. Clique is hard to approximate within $\mathrm{n}^{1\text{-epsilon}}$. In *Proc.* 37^{th} *IEEE FOCS*, pages 627–636, 1996.

9 J. Håstad. Some optimal inapproximability results. *J. ACM*, 48(4):798–859, 2001.

10 S. Khot. Ruling out PTAS for graph min-bisection, dense k-subgraph, and bipartite clique. *SIAM J. Comput.*, 36(4):1025–1071, 2006.

11 J. Kleinberg, C. Papadimitriou, and P. Raghavan. A microeconomic view of data mining. *Data Min. Knowl. Discov.*, 2(4):311–324, 1998.

12 M. Mitzenmacher. On the hardness of finding optimal multiple preset dictionaries. *IEEE Transactions on Information Theory*, 50(7):1536–1539, 2004.

13 R. Raz. A parallel repetition theorem. *SIAM J. Comput.*, 27(3):763–803, 1998.

14 R. Rubinfeld and M. Sudan. Robust characterizations of polynomials with applications to program testing. *SIAM J. Comput.*, 25(2):252–271, 1996.

15 S. Sahni. Approximate algorithms for the 0/1 knapsack problem. *J. ACM*, 22(1):115–124, 1975.

16 Y. Yubo and X. Chengxian. Improved randomized algorithm for the equivalent 2-catalog segmentation problem. *Numerical Mathematics*, 14(2):128–135, 2005.

Determining the Winner of a Dodgson Election is Hard*

Michael Fellows[1], Bart M. P. Jansen[2], Daniel Lokshtanov[3], Frances A. Rosamond[1], and Saket Saurabh[4]

1 **Parameterized Complexity Research Unit, University of Newcastle**
Callaghan, NSW Australia
{Michael.Fellows|Frances.Rosamond}@newcastle.edu.au

2 **Department of Information and Computing Sciences, Utrecht University**
P.O. Box 80.089, 3508 TB, Utrecht, The Netherlands
bart@cs.uu.nl

3 **Department of Informatics, University of Bergen**
N-5020 Bergen, Norway
daniello@ii.uib.no

4 **Institute of Mathematical Sciences**
CIT Campus, Taramani, 600 113 Chennai, India
saket@imsc.res.in

Abstract

Computing the *Dodgson Score* of a candidate in an election is a hard computational problem, which has been analyzed using classical and parameterized analysis. In this paper we resolve two open problems regarding the parameterized complexity of Dodgson Score. We show that Dodgson Score parameterized by the target score value k does not have a polynomial kernel unless the polynomial hierarchy collapses to the third level; this complements a result of Fellows, Rosamond and Slinko who obtain a non-trivial kernel of exponential size for a generalization of this problem. We also prove that Dodgson Score parameterized by the number n of votes is hard for $W[1]$.

1998 ACM Subject Classification F.2.2.

Keywords and phrases Dodgson Score, Parameterized Complexity, Kernelization Lower Bounds

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.459

1 Introduction

Complexity issues play an important role in the relatively new area of computational social choice, especially in the area of election systems, which has applications in finance and economics (agreement on the winner of an auction), internet search engines (agreement on the order of web pages presented), web mining (consensus is the notion of "public opinion"), mechanism design (agreement by participants in large networks involving autonomous software agents), and computational biology (finding consensus in feature selection), among many others [1, 14, 15]. The involvement of increasingly larger numbers of participants and the increasing sophistication of the information objects of debate, have made election systems a vital area of computer science research.

* Bart Jansen was supported by the Netherlands Organisation for Scientific Research (NWO), project "KERNELS: Combinatorial Analysis of Data Reduction".

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 459–468

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

In this paper we study the hard election problem Dodgson Score. We consider an election in which we allow each voter to specify a complete preference ranking of the candidates: each vote is a strict total order on the set of candidates, and a vote in an election with three candidates could be represented as $a < b < c$ stating that candidate a is least preferred and c is most preferred. Given the votes that were cast in an election, we can compare the relative ranking of two candidates a, b as follows: candidate a beats candidate b in pairwise comparison if a is ranked above b more often than below b. A candidate who beats every other candidate in pairwise comparison is said to be a *Condorcet winner*. If such a winner exists then it must be unique, and it wins the election. But unfortunately a Condorcet winner may not always exist, as is shown by the following election with three candidates and three voters: $a < b < c, b < c < a, c < a < b$. This situation has a cyclic preference structure: candidate a beats b, candidate b beats c and c beats a (in pairwise comparison), so there is no candidate who beats all others. In 1876 the mathematician Charles Dodgson formulated a rule that defines the winner of an election even if there is no Condorcet winner [11, 6]. The idea is to measure how close a candidate is to being a Condorcet winner; the candidate who is closest then wins the election. This can be formalized as follows. The *Dodgson score* of a candidate c in an election, is defined to be the minimum number of swaps of adjacent candidates in the voter's preference orders that have to be made to ensure that c becomes a Condorcet winner. The candidates that have the minimum Dodgson score are the winners of the election. Dodgson's rule is not the only voting scheme resulting from Condorcet's criterion; similar schemes have been suggested by Young and Kemeny [22].

Unfortunately, Dodgson Score, Young Score and Kemeny Score and many other election problems are NP-hard or worse, and finding an "approximate" winner of an election is hard [10] and usually not appropriate. Thus, election problems are well-suited for parameterized analysis because it offers an exact result, taking advantage of natural parameters to the problems, such as the number of votes that were cast, the number of candidates, or the score of a candidate.

1.1 Earlier Work

Bartholdi et al. initiated the study of the complexity of the Dodgson voting scheme in 1989 [2], when they showed that determining the winner of a Dodgson election is NP-hard. They also proved that computing the Dodgson or Kemeny score of a given candidate is NP-complete. The complexity of the winner problem for Dodgson elections was later established exactly; Hemaspaandra et al. [20] showed in 1997 that this problem is complete for $\mathrm{P}_{||}^{\mathrm{NP}}$ ("parallel access to NP").

McCabe-Dansted [21] was the first to investigate Dodgson Score using the framework of parameterized complexity, and observed that the ILP formulation of the problem from Bartholdi et al. [2] implies fixed-parameter tractability for the parameterization by the number m of candidates in the election. The parameterization by the target score k was first studied in 2007, when Fellows and Rosamond showed that k-Dodgson Score is in FPT. The group of Betzler et al. [4, 5] independently reached the same conclusion and obtained a dynamic programming algorithm with running time $O(2^k \cdot nk + nm)$ where n is the number of votes and m the number of candidates. Fellows, Rosamond and Slinko [16] considered a generalization of Dodgson's rule where each possible preference ranking specifies a cost for every swap that can be made; a candidate wins the election if the minimum total cost of making that candidate a Condorcet winner is not higher than the minimum cost of making any other candidate a Condorcet winner. They obtained a kernel of exponential size $O(e^{O(k^2)})$ for this k-Generalized Dodgson Score problem.

The election problems Kemeny Score and Young Score have also been studied from the parameterized perspective. The Young Score problem is $W[2]$-complete when parameterized by the target score, and the same holds when using the dual of this parameter [4, 5]. The Kemeny Score problem admits several natural parameterizations that lead to fixed-parameter tractability. Results have been found for parameters 'number of votes', 'average Kendall-Tau distance', 'maximum range (or maximum Kendall-Tau distance)' and combined parameters of 'number of votes and average KT-distance', and 'number of votes and maximum KT-distance' [3].

1.2 Our Results

The parameterized analysis of Dodgson Score by Betzler et al. [5] left two open problems unanswered: 1) does k-Dodgson Score admit a polynomial kernel when parameterized by the target score, and 2) is the problem fixed-parameter tractable when parameterized by the number of votes? We answer both questions in this paper. We use the framework developed by Bodlaender et al. [7] in combination with a theorem by Fortnow and Santhanam [18] to prove that there is no polynomial kernel for k-Dodgson Score unless the polynomial hierarchy collapses to the third level (denoted as $PH = \Sigma_3^p$), and further [9]. Our second result is a non-trivial reduction establishing that k-Dodgson Score parameterized by the number of votes is hard for $W[1]$.

2 Preliminaries

In this section we formalize some notions that were introduced in Section 1. An election is a tuple (V, C) where V is a multiset of votes, and C is a set of candidates. A vote $v \in V$ is a preference list on the candidates, i.e. a strict total ordering. For candidates $a, b \in C$ the value $n_{a,b}$ counts the number of votes in V that rank a above b. A Condorcet winner is a candidate $x \in C$ such that $n_{x,y} > n_{y,x}$ for all $y \in C \setminus \{x\}$. To swap candidate x upwards in a vote $v \in V$ means to exchange the positions of x and the candidate immediately above it in the ranking; an upward swap operation is undefined if x is already the most preferred candidate in the vote. For example, if $x < z < w < y$ is a vote, then swapping x upwards once results in the vote $z < x < w < y$. We say that the candidate x *gains a vote* on candidate z through this swap, since this swap increases $n_{x,z}$ by one and decreases $n_{z,x}$ by one. The *Dodgson score* of a candidate $x \in C$ is the minimum number of swaps needed to make x a Condorcet winner. It is not hard to verify that if x can be made a Condorcet winner by k swaps, then this can also be done by k swaps that only move candidate x upwards. Consult [21, Lemma 4.0.5] for a formal proof of this claim. Therefore we may also define the Dodgson score as the minimum number of *upwards* swaps of x that are required to make x a Condorcet winner.

The theory of parameterized complexity [13] offers a toolkit for the theoretical analysis of the structure of NP-hard problems. A parameterized decision problem is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where an instance (x, k) is composed of the classical input x and the parameter value k that describes some property of x. A parameterized problem L is in the class (strongly uniform) FPT (for Fixed-Parameter Tractable) if there is an algorithm that decides L in $f(k)p(|x|)$ time, where p is a polynomial and f is a computable function. A *kernelization algorithm* (kernel) [23] is a mapping that transforms an instance $(x, k) \in \Sigma^* \times \mathbb{N}$ in $p(|x| + k)$ time for some polynomial p, into an equivalent instance (x', k') such that $(x, k) \in L \Leftrightarrow (x', k') \in L$ and such that $|x'|, k' \leq f(k)$ for some computable function f. The function f is called the size of the kernel. Recent developments in the theory of kernelization

have yielded tools to show that certain problems are unlikely to have kernels of polynomial size [7]. The Dodgson Score problem is formally defined as follows:

Dodgson Score
Instance: A set C of candidates, a distinguished candidate $x \in C$, a multiset V of votes and a positive integer k.
Question: Can x be made a Condorcet winner by making at most k swaps between adjacent candidates?

We consider two different parameterizations in this work. When the problem is parameterized by the number of allowed swaps k then we will refer to it as k-Dodgson Score; the other variant considers a bounded number of votes $n := |V|$ which we call n-Dodgson Score.

3 Kernelization Lower Bound for k-Dodgson Score

In this section we prove that k-Dodgson Score does not have a polynomial kernel unless $PH = \Sigma_3^p$. To prove this result we need some notions related to parameterized reducibility.

▶ **Definition 1** ([8]). Let P and Q be parameterized problems. We say that P is polynomial parameter reducible to Q, written $P \leq_{Ptp} Q$, if there exists a polynomial time computable function $g : \Sigma^* \times \mathbb{N} \to \Sigma^* \times \mathbb{N}$ and a polynomial p, such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ (a) $(x, k) \in P \Leftrightarrow (x', k') = g(x, k) \in Q$ and (b) $k' \leq p(k)$. The function g is called *polynomial parameter transformation.*

▶ **Theorem 2** ([8]). *Let P and Q be parameterized problems and $\tilde{P}$ and $\tilde{Q}$ be the unparameterized versions of P and Q respectively. Suppose that $\tilde{P}$ is NP-hard and $\tilde{Q}$ is in NP. Furthermore if there is a polynomial parameter transformation from P to Q, then if Q has a polynomial kernel then P also has a polynomial kernel.*

We use the following problem as the starting point for our transformation:

Small Universe Set Cover
Instance: A set family $\mathcal{F} \subseteq 2^U$ of subsets of a finite universe U and a positive integer $k \leq |\mathcal{F}|$.
Question: Is there a subfamily $\mathcal{F}' \subseteq \mathcal{F}$ with $|\mathcal{F}'| \leq k$ such that $\cup_{S \in \mathcal{F}'} S = U$?
Parameter: The value $k + |U|$.

Small Universe Set Cover is a parameterized version of the NP-complete Set Cover problem [19, SP5]. We need the following incompressibility result for this problem [12, Theorem 2]:

▶ **Theorem 3.** *The problem* Small Universe Set Cover *parameterized by $k + |U|$ does not admit a polynomial kernel unless $PH = \Sigma_3^p$.*

The transformation that we shall use to prove that k-Dodgson Score does not have a polynomial kernel (unless $PH = \Sigma_3^p$) is similar in spirit to the reduction from Exact Cover by 3-sets which was originally used to show that Dodgson Score is NP-complete [2]. Let $(U, \mathcal{F}, k)$ be an instance of Small Universe Set Cover. We show how to construct an equivalent instance (V, C, x, k') of k-Dodgson Score with $k' := k(|U| + 1)$ in polynomial time. Since the problem can be solved in polynomial time when $|\mathcal{F}| < 3$, we may assume without loss of generality that the set family $\mathcal{F}$ contains at least 3 sets.

The set of candidates is composed of several parts. We create one candidate for each element u in the universe U; we will use an element $u \in U$ to refer both to the corresponding

candidate and to the element of the finite universe, since the meaning will be clear from the context. We also take one candidate x to use as the distinguished candidate for whom the Dodgson score must be computed, one candidate y that will encode the fact that we must cover the universe with exactly k subsets, and finally we use three sets of dummy candidates D_0, D_1 and D_2 that are needed for padding. These dummy candidates will ensure that we can make the distance between x and y in the total orders sufficiently large, i.e. that it takes a lot of swaps for x to gain a vote on y. We want to ensure that x beats all the dummy candidates in pairwise comparison in the initial situation, to ensure that the dummies do not interfere with the encoding of the set cover instance. Our three sets of dummies D_0, D_1 and D_2 each contain $|U|+1$ candidates. If we want to use some $d \leq |U|+1$ dummy candidates that rank above x in the i-th vote that we create, then we use d candidates from the set $D_{i \bmod 3}$; the other dummies are ranked below x. Since x beats every dummy candidate in at least two out of three votes, this ensures that x will beat all dummy candidates in pairwise comparison if we use at least 5 votes. Using this scheme we will from now on write D^j to denote a set of $j \leq |U|+1$ dummy candidates that can be used in the vote we are constructing.

The set V of votes is built out of two parts, each containing $|\mathcal{F}|$ votes. Since $|\mathcal{F}| \geq 3$ this will ensure that we create at least 6 votes. Using the terminology of [2] we create a set of *swing* votes corresponding to elements of $\mathcal{F}$, and a set of *equalizing* votes that create the proper initial conditions. We introduce an abbreviation to write down total orders: if $C' \subseteq C$ is a set of candidates, then by writing a total order $a < C' < b$ we mean a total order in which all candidates of C' are ranked below b and above a. The relative ranking of the candidates in C' among each other is not important. We now define the two parts of the vote set.

Swing votes. For every $S \in \mathcal{F}$ we make a vote $(\ldots < x < S < D^{|U|-|S|} < y)$. All candidates that are not explicitly mentioned in the construction are ranked below x in arbitrary order. The set S in this vote represents the candidates corresponding to the universe elements in $S \subseteq U$.

The swing votes correspond to the sets in the family $\mathcal{F}$. Observe that it takes exactly $|U|+1$ switches for x to gain a swing vote on y. The name “swing vote" comes from the fact that if x can become a Condorcet winner in $k(|U|+1)$ switches, then all those switches must be made in swing votes.

Equalizing votes. The goal of the set of equalizing votes is to create initial conditions in which x must gain k votes on candidate y, and one vote on every candidate corresponding to some $u \in U$ in order to become the Condorcet winner. We construct the equalizing votes so that no switches made in them will allow x to become a winner in $k(|U|+1)$ steps. We do not give an explicit construction for the equalizing votes; instead we present the conditions that they must satisfy. It will be easy to see that such a set of votes exists, and can be constructed in polynomial time.

1. For every candidate corresponding to an element $u \in U$, the number of *equalizing* votes in which x is ranked above u is equal to the number of *swing* votes in which x is ranked below u. This ensures that overall, every candidate u is ranked above x exactly as often as below x; hence x needs to gain one vote on every u to beat it in pairwise comparison.
2. There are $|\mathcal{F}|-k+1$ equalizing votes in which x is ranked above y. Since x is ranked below y in all swing votes, this implies that there are $|\mathcal{F}|-k+1$ votes in which x ranks above y, and $2|\mathcal{F}|-(|\mathcal{F}|-k+1) = |\mathcal{F}|+k-1$ votes in which x ranks below y. The reader may verify that this means that x needs to gain at least k votes on y to beat y in a pairwise comparison.

3. Whenever x is ranked below y, then (by inserting dummies if necessary) there are at least $|U|+1$ candidates between x and y. This ensures that at least $|U|+2$ swaps are needed for x to gain an equalizing vote on y.

This concludes the construction of the instance (V, C, x, k').

▶ **Lemma 4.** *If the instance $(U, \mathcal{F}, k)$ has a set cover of size k, then candidate x can be made a Condorcet winner in the election (V, C, x, k') by k' swaps.*

Proof. Suppose $\mathcal{F}' \subseteq \mathcal{F}$ is a set cover of size k. Every $S \in \mathcal{F}'$ corresponds to a swing vote. Consider the effect of swapping x upwards for $|U|+1$ steps in the swing votes corresponding to the elements of $\mathcal{F}'$. Since there are exactly $|U|$ candidates between x and y in every swing vote, this means that x gains these k votes on y. Since $\mathcal{F}'$ is a set cover of U it follows from the construction of the swing votes that we must have swapped x over every candidate $u \in U$ at least once. By the earlier observations this shows that after these $k' = k(|U|+1)$ swaps the candidate x must be a Condorcet winner. ◀

▶ **Lemma 5.** *If candidate x can be made a Condorcet winner in the election (V, C, x, k') by k' swaps, then instance $(U, \mathcal{F}, k)$ has a set cover of size k.*

Proof. Assume there is some series of $k(|U|+1)$ swaps that makes x a Condorcet winner. By the observations in the preliminaries we may assume that these swaps only move x upwards. Since x needs to gain k votes on y in order to become a Condorcet winner, we can conclude that at most $|U|+1$ swaps on average can be used for every vote that x gains over y. But by construction it is impossible to improve over y using fewer than $|U|+1$ swaps per vote, which shows that none of the swaps can be made in *equalizing votes* since there it takes at least $|U|+2$ swaps for x to improve over y. It follows that the swaps that make x a Condorcet winner in $k(|U|+1)$ steps must be composed of $|U|+1$ swaps in k different swing votes. Since x had to gain one vote on every candidate corresponding to $u \in U$ in order to become a Condorcet winner, we may conclude that in these k swing votes every candidate $u \in U$ was ranked above x at least once. But this shows that the sets corresponding to the k swing votes form a set cover for U of size k, which shows that U has a set cover of the requested size. ◀

It is not hard to verify that the transformation can be computed in polynomial time. The transformation is a polynomial parameter transformation because the parameter $k' = k(|U|+1)$ of the k-Dodgson Score instance is bounded by the square of the original parameter $k+|U|$. By combining Theorem 2 with Theorem 3 the existence of this polynomial parameter transformation yields the following theorem.

▶ **Theorem 6.** *k-Dodgson Score does not admit a polynomial kernel unless $PH = \Sigma_3^p$.*

4 Parameterized Hardness of n-Dodgson Score

We now consider the parameterization by the number of votes n and show that this leads to $W[1]$-hardness. We use a reduction from the following well-known problem [17].

Multi-Colored Clique
Instance: A simple undirected graph $G = (V, E)$, a positive integer k and a coloring function $c : V \to \{1, 2, \ldots, k\}$ on the vertices.
Question: Is there a clique in G that contains exactly one vertex from each color class?
Parameter: The value k.

We give a FPT-reduction from Multi-Colored Clique to n-Dodgson Score. In particular, given an instance $(G = (V, E), c, k)$ of Multi-Colored Clique we construct an instance (C', V', x', k') of n-Dodgson Score such that $|V'| = n = 4(\binom{k}{2} + k)$.

Let $V_1, \ldots, V_k$ be the color classes of G, that is, for every $v \in V_i$ we have $c(v) = i$. For every pair of distinct integers $1 \leq i < j \leq k$ we define $E_{i,j}$ to be the set of edges with one endpoint in V_i and one in V_j. We will assume without loss of generality that all color classes of G have the same number N of vertices, and that between every pair of color classes there are exactly M edges. We define the target score value k' of the Dodgson Score instance as $k' := ((N+1)(Mk+1)+2)k + (5M-3)\binom{k}{2}$. The set C' of candidates is built out of five groups.

1. We have a distinguished candidate x' for which we need to compute the Dodgson score.
2. We use $3k'$ dummy candidates, just as in the proof of Theorem 6. This allows us to use up to k' dummy candidates in each vote, while maintaining the property that the candidate x' is ranked above every dummy candidate in more than half of the votes.
3. For every color class $1 \leq i \leq N$ there are candidates a_i^p for $0 \leq p \leq N+2$.
4. For every pair of color classes $1 \leq i < j \leq k$ there are candidates $a_{i,j}^p$ for $1 \leq p \leq M+1$.
5. For every edge $e \in E_{i,j}$ there are candidates e_i, e_i', e_j and e_j'.

From these definitions it is easy to verify that the number of candidates is polynomial in the size of the Multi-Colored Clique instance. We now describe the vote set. As in the proof of Theorem 6 we will distinguish between *swing votes* and *equalizing votes*. There are $2(\binom{k}{2} + k)$ votes of each type, and hence $|V'| = n = 4(\binom{k}{2} + k)$ from which it follows that the parameter n for the n-Dodgson Score instance is polynomial in the parameter k of the Multi-Colored Clique instance.

Equalizing votes. The equalizing votes create the right initial conditions for the election. We build the equalizing votes such that in the resulting election the distinguished candidate x' must gain exactly one vote on each non-dummy candidate in order to win the election. We ensure that no swaps made in an equalizing vote can allow x' to become a Condorcet winner in k' steps, by ranking k' dummy candidates immediately above x' in every equalizing vote. It is not hard to see that we do not need more equalizing votes than swing votes to encode these requirements.

Swing votes. The swing votes encode the behavior of the Multi-Colored Clique instance into the election. Every edge e gets an identification number $\mathrm{ID}(e)$ between 1 and M. Since the total number of edges is $M\binom{k}{2}$ the identification of two edges may be the same, but we ensure that for two distinct edges e^1, e^2 both in $E_{i,j}$ we always have $\mathrm{ID}(e^1) \neq \mathrm{ID}(e^2)$. Similarly we give every vertex $v \in V$ an identification number $\mathrm{ID}(v)$ between 1 and N, and we ensure that distinct vertices in the same color class have different ID's. As in the previous construction we know that only the part of the vote above x' is relevant, so we do not show the remainder. None of the described candidates are dummies, unless specified otherwise. For every pair of integers $1 \leq i < j \leq k$ we make two swing votes, $v_{i,j}^1$ and $v_{i,j}^2$ as follows.

$$v_{i,j}^1 : \quad x < a_{i,j}^1 < \ldots < a_{i,j}^2 < \ldots < a_{i,j}^3 < (\ldots) < a_{i,j}^{M+1} \tag{1}$$

$$v_{i,j}^2 : \quad x < a_{i,j}^{M+1} < \ldots < a_{i,j}^M < \ldots < a_{i,j}^{M-1} < (\ldots) < a_{i,j}^1 \tag{2}$$

The gaps between consecutive candidates $a_{i,j}^p$ are filled as follows. For every edge $e \in E_{i,j}$ we insert e_i, e_i', e_j and e_j' between $a_{i,j}^{\mathrm{ID}(e)}$ and $a_{i,j}^{\mathrm{ID}(e)+1}$ in $v_{i,j}^1$. Also, we insert e_i, e_i', e_j and e_j' between $a_{i,j}^{\mathrm{ID}(e)+1}$ and $a_{i,j}^{\mathrm{ID}(e)}$ in $v_{i,j}^2$. Notice that between any consecutive $a_{i,j}^p$'s in $v_{i,j}^1$ and $v_{i,j}^2$ there are exactly 4 other candidates.

For every integer $1 \leq i \leq k$ we make two swing votes, v_i^1 and v_i^2 as follows.

$$v_i^1 : \quad x < a_i^0 < \ldots < a_i^1 < \ldots < a_i^2 < \ldots < a_i^3 < (\ldots) < a_i^N \tag{3}$$

$$v_i^2 : \quad x < a_i^{N+2} < \ldots < a_i^{N+1} < \ldots < a_i^N < \ldots < a_i^{N-1} < (\ldots) < a_i^2 \tag{4}$$

For every edge e with one endpoint v in V_i we add e_i between $a_i^{\mathrm{ID}(v)-1}$ and $a_i^{\mathrm{ID}(v)}$ in v_i^1 and we add e_i' between $a_i^{\mathrm{ID}(v)+2}$ and $a_i^{\mathrm{ID}(v)+1}$ in v_i^2. Having done this for every edge, we add dummy candidates between each consecutive pair of a_i^p's in v_i^1 and v_i^2 such that the total number of candidates between each consecutive pair of a_i^p's in v_i^1 and v_i^2 is exactly kM. This concludes the construction of (C', V', x', k').

▶ **Lemma 7.** *If G contains a colored k-clique, then x' can be made a winner of the election (C', V', x', k') in $k' = ((N+1)(kM+1)+2)k + (5M-3)\binom{k}{2}$ swaps.*

Proof. Let $C = c_1, c_2 \ldots c_k$ be a clique in G such that $c_i \in V_i$. For each vertex $c_i \in C$ we move x' in v_i^1 such that x beats $a_i^{\mathrm{ID}(c_i)}$. We also move x' in v_i^2 such that x beats $a_i^{\mathrm{ID}(c_i)+1}$. For each value of i this takes exactly $(N+1)(kM+1)+2$ swaps: we need 1 swap to move over a_i^0 in v_i^1 and 1 swap to move over a_i^{N+2} in v_i^2, and for all $N+1$ other candidates a_i^p we need to swap over the block of kM in front of them and over the candidates themselves, resulting in $(N+1)(kM+1)$ more swaps. Thus in total there are $((N+1)(kM+1)+2)k$ swaps in the v_i^1 and v_i^2 swing votes.

For every pair of distinct integers $1 \leq i < j \leq k$ we move x' in $v_{i,j}^1$ such that x beats $a_{i,j}^{\mathrm{ID}(c_ic_j)}$, and move x' in $v_{i,j}^2$ such that x beats $a_{i,j}^{\mathrm{ID}(c_ic_j)+1}$. The number of swaps to do this is $(5M-3)\binom{k}{2}$. Thus the total number of swaps is $((N+1)(kM+1)+2)k + (5M-3)\binom{k}{2} = k'$.

We show that x has gained a swing vote on each non-dummy candidate. It is easy to see that for every $1 \leq i \leq k$, the candidate x' has gained a swing vote on all a_i^p's and that for every $1 \leq i < j \leq k$, x' has gained a swing vote on all $a_{i,j}^p$'s. Observe that in the two swing votes $v_{i,j}^1$ and $v_{i,j}^2$, x' has gained a swing vote on all candidates e_i, e_i', e_j and e_j' *except* for the four candidates corresponding to the edge $e[i,j] = c_ic_j$. Let these four candidates be $e_i[i,j]$, $e_i'[i,j]$, $e_j[i,j]$ and $e_j'[i,j]$ respectively. However, x' gains a swing vote on $e_i[i,j]$ in v_i^1, on $e_i'[i,j]$ in v_i^2, on $e_j[i,j]$ in v_j^1 and on $e_j'[i,j]$ in v_j^2. This concludes the proof of the lemma. ◀

▶ **Lemma 8.** *If x' can be made a winner of the election (C', V', x', k') in $k' = ((N+1)(kM+1)+2)k + (5M-3)\binom{k}{2}$ swaps, then G contains a colored k-clique.*

Proof. Observe that for any fixed i we need to perform at least $(N+1)(kM+1)+2$ swaps in v_i^1 and v_i^2 in total in order to gain a swing vote on all a_i^p's. Similarly for any fixed $1 \leq i < j \leq k$ we need to perform at least $5M-3$ swaps in $v_{i,j}^1$ and $v_{i,j}^2$ in total, in order to gain a swing vote on all $a_{i,j}^p$'s.

Thus, if strictly more than $(N+1)(kM+1)+2$ swaps are performed in v_i^1 and v_i^2 in total for some i, or if more than $5M-3$ swaps are performed in $v_{i,j}^1$ and $v_{i,j}^2$ in total for some i, j, then the total number of swaps performed must be greater than k' if the swaps make x' a Condorcet winner. So under the given assumptions for each i, exactly $(N+1)(kM+1)+2$ swaps are performed in v_i^1 and v_i^2 in total, and for each $1 \leq i < j \leq k$, exactly $5M-3$ swaps are performed in $v_{i,j}^1$ and $v_{i,j}^2$ in total.

For a fixed i, if x' has gained a swing vote for all the a_i^p's in v_i^1 and v_i^2, then there must be some $c_i \in V_i$ such that x' has been moved right over $a_i^{\mathrm{ID}(c_i)}$ in v_i^1 and right over $a_i^{\mathrm{ID}(c_i)+1}$ in v_i^2. Similarly, for every $1 \leq i < j \leq k$ there must be some $e[i,j] \in E_{i,j}$ such that x' has been moved right over $a_{i,j}^{\mathrm{ID}(e[i,j])}$ in $v_{i,j}^1$ and moved right over $a_{i,j}^{\mathrm{ID}(e[i,j])+1}$ in $v_{i,j}^2$. Notice that x' did not get any swing vote on $e_i[i,j]$, $e_i'[i,j]$, $e_j[i,j]$ and $e_j'[i,j]$. The only other places x' could

have gotten a swing vote on these candidates are in v_i^1, v_i^2, v_j^1 and v_j^2 respectively. We prove that $e[i,j]$ is incident to c_i. Let v_i be the vertex incident to $e[i,j]$ in V_i. If $\mathrm{ID}(c_i) < \mathrm{ID}(v_i)$ then x' does not gain a swing vote on $e[i,j]_i$. Hence $\mathrm{ID}(c_i) \geq \mathrm{ID}(v_i)$. If, on the other hand, $\mathrm{ID}(c_i) > \mathrm{ID}(v_i)$ then x' does not gain a swing vote on $e'[i,j]_i$, and therefore we must have $\mathrm{ID}(c_i) \leq \mathrm{ID}(v_i)$. But then $v_i = c_i$ and therefore we know that for each i the vertex c_i that is selected in the votes v_i^1 and v_i^2 is an endpoint of the edge that was selected in the votes $v_{i,j}^1$ and $v_{i,j}^2$. Using $e[i,j]_j$ and $e'[i,j]_j$ one can similarly prove that $e[i,j]$ is incident to c_j. Hence $C = \{c_1, c_2, \ldots, c_k\}$ forms a clique in G. This concludes the proof of the lemma. ◀

The construction of (C', V', x', k') together with Lemmata 7 and 8 shows that there is a FPT-reduction from Multi-Colored Clique to n-Dodgson Score. Since it is well-known [17] that Multi-Colored Clique is hard for $W[1]$, we obtain the following result.

▶ **Theorem 9.** *n-Dodgson Score is hard for $W[1]$.*

5 Conclusions and Discussion

In this paper we answered two open problems with respect to the parameterized complexity of Dodgson Score. The parameterization k-Dodgson Score does not admit a polynomial kernel unless the polynomial hierarchy collapses, and n-Dodgson Score is hard for $W[1]$. The proof that k-Dodgson Score does not have a polynomial kernel unless $PH = \Sigma_3^p$ also implies that the exponential size kernel by Fellows et al. [16] for k-Generalized Dodgson Score cannot be improved to a polynomial kernel unless $PH = \Sigma_3^p$.

In a natural variant of the Dodgson Score problem we are given a set of votes over a set of candidates C, together with an integer k, and asked whether any candidate can be made a Condorcet Winner by performing at most k swaps. A simple construction extends the hardness result of Theorems 6 and 9 to this problem as well.

References

1 Nir Ailon, Moses Charikar, and Alantha Newman. Aggregating inconsistent information: Ranking and clustering. *J. ACM*, 55(5), 2008.

2 J. J. Bartholdi, C. A. Tovey, and M. A. Trick. Voting schemes for which it can be difficult to tell who won the election. *Social Choice and Welfare*, 6(2):157–165, 1989.

3 Nadja Betzler, Michael R. Fellows, Jiong Guo, Rolf Niedermeier, and Frances A. Rosamond. Fixed-parameter algorithms for Kemeny rankings. *Theor. Comput. Sci.*, 410(45):4554–4570, 2009.

4 Nadja Betzler, Jiong Guo, and Rolf Niedermeier. Parameterized computational complexity of Dodgson and Young elections. In *Proc. 11th SWAT*, pages 402–413, 2008.

5 Nadja Betzler, Jiong Guo, and Rolf Niedermeier. Parameterized computational complexity of Dodgson and Young elections. *Information and Computation*, 208(2):165–177, 2010.

6 Duncan Black, Robert Albert Newing, Iain McLean, Alistair McMillan, and Burt L. Monroe. *The theory of committees and elections*. Springer, 1998.

7 Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.

8 Hans L. Bodlaender, Stéphan Thomassé, and Anders Yeo. Kernel bounds for disjoint cycles and disjoint paths. In *Proc. 17th ESA*, pages 635–646, 2009.

9 Jin-yi Cai, Venkatesan T. Chakaravarthy, Lane A. Hemaspaandra, and Mitsunori Ogihara. Competing provers yield improved Karp-Lipton collapse results. *Inf. Comput.*, 198(1):1–23, 2005.

10 Ioannis Caragiannis, Jason A. Covey, Michal Feldman, Christopher M. Homan, Christos Kaklamanis, Nikos Karanikolas, Ariel D. Procaccia, and Jeffrey S. Rosenschein. On the approximability of Dodgson and Young elections. In *Proc. 20th SODA*, pages 1058–1067, 2009.

11 C. L. Dodgson. *A method for taking votes on more than two issues.* Clarendon Press, Oxford, 1876.

12 Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Incompressibility through colors and IDs. In *Proc. 36th ICALP*, pages 378–389, 2009.

13 Rod Downey and Michael R. Fellows. *Parameterized Complexity.* Monographs in computer science. Springer, New York, 1999.

14 Cynthia Dwork, Ravi Kumar, Moni Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW*, pages 613–622, 2001.

15 Ronald Fagin, Ravi Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *SIGMOD Conference*, pages 301–312, 2003.

16 Michael Fellows, Frances Rosamond, and Arkadii Slinko. Sensing God's will is fixed parameter tractable. Technical report, ResearchSpace@Auckland 561, 2008.

17 Michael R. Fellows, Danny Hermelin, Frances A. Rosamond, and Stéphane Vialette. On the parameterized complexity of multiple-interval graph problems. *Theor. Comput. Sci.*, 410(1):53–61, 2009.

18 Lance Fortnow and Rahul Santhanam. Infeasibility of instance compression and succinct PCPs for NP. In *Proc. 40th STOC*, pages 133–142, 2008.

19 Michael R. Garey and David S. Johnson. *Computers and Intractability, A Guide to the Theory of NP-Completeness.* W.H. Freeman and Company, New York, 1979.

20 Edith Hemaspaandra, Lane A. Hemaspaandra, and Jörg Rothe. Exact analysis of Dodgson elections: Lewis Carroll's 1876 voting system is complete for parallel access to NP. *J. ACM*, 44(6):806–825, 1997.

21 John C. McCabe-Dansted. Approximability and computational feasibility of Dodgson's rule. Master's thesis, University of Auckland, 2006.

22 I. McLean and A. Urken. *Classics of Social Choice.* University of Michigan Press, Ann Arbor, Michigan, 1995.

23 Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms.* Oxford University Press, 2006.

Verifying Recursive Active Documents with Positive Data Tree Rewriting

Blaise Genest[1,2], Anca Muscholl[3], and Zhilin Wu[4]

1 **CNRS, IPAL UMI, joint with I2R-A*STAR-NUS, Singapore**
2 **CNRS, IRISA UMR, joint with Université Rennes I, France**
bgenest@irisa.fr
3 **LaBRI, Université Bordeaux/CNRS, France**
anca@labri.fr
4 **LaBRI, Université Bordeaux/CNRS, France**
zlwu@labri.fr

Abstract

This paper considers a tree-rewriting framework for modeling documents evolving through service calls. We focus on the automatic verification of properties of documents that may contain data from an infinite domain. We establish the boundaries of decidability: while verifying documents with recursive calls is undecidable, we obtain decidability as soon as either documents are in the *positive-bounded* fragment (while calls are unrestricted), or when there is a bound on the number of service calls (bounded model-checking of unrestricted documents). In the latter case, the complexity is NexpTime-complete. Our data tree-rewriting framework resembles Guarded Active XML, a platform handling XML repositories that evolve through web services. The model here captures the basic features of Guarded Active XML and extends it by node renaming and subtree deletion.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.469

1 Introduction

From static in house solutions, databases have become more and more open to the world, offering e.g. half-open access through web services. As usual for open systems, their design requires a careful static analysis process, helping to guarantee that no malicious client may take advantage of the system in a way for which the system was not designed. Static analysis of such systems very recently brought together two areas - databases, with emphasis on semi-structured XML data, and automated verification, with emphasis on model-checking infinite-state systems. Systems modeling dynamical evolution of data are pretty challenging for automated verification, as they involve feedback loops between semi-structured data, possibly with values from unbounded domains, and the workflow of services. If both topics have been studied extensively on its own, very few papers tackle decidability of algorithms when all aspects are present at the same time.

An interesting model emerged recently for handling XML repositories evolving through web services, namely Active XML (AXML) [4]. These are XML documents that evolve dynamically, containing implicit data in form of embedded service calls. Services may be recursive, so the evolution of such documents is both non-deterministic and unbounded in time. A first paper analyzing the evolution of AXML documents considered *monotonous* documents [3]. With this restriction, as soon as a service is enabled in a document, then from this point on the service cannot be disabled and calling it can only extend the document. In particular, information can never be deleted. Recently, a workflow-oriented version of AXML was proposed in [5]: the *Guarded AXML* model (GAXML for short) adds guards

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 469–480

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to service calls, thus controlling the possible evolution of active documents. Decidability in co-2NexpTime of static analysis for the *recursion-free GAXML* fragment w.r.t. a variant of LTL with data tree patterns as atomic formulas was established in [5]. Static analysis is more complex in [5], due to the presence of unbounded data. The crucial restriction needed for decidability is a uniform bound on the number of possible service calls. Compared to [3], service invocation can terminate, and more importantly, negative guards can be used. But still, deletion of data is not possible. Finally, the GAXML model relies on a rather involved semantics of service calls.

In this work, our aim is twofold. First, we aim at embedding and extending the GAXML model in a simpler framework based on tree rewriting. Our model DTPRS (*data tree pattern rewriting systems*) uses the same basic ingredients as GAXML, which are tree patterns for guards and queries. However, our formalism allows to describe several possible effects of a service call: materialization of implicit data like in GAXML, but also deletion and modification of existing document parts. This model is a simplified version of the TPRS model proposed in [15], but in this setting it can additionally handle unbounded data.

Our second, and main objective is to get decidability of static analysis of DTPRS without relying on a bound on the number of service calls. For doing that, we use a technique that emerged in the verification of particular infinite-state systems such as Petri nets and lossy channel systems. The main concept is known in verification as *well-structured transition systems* (WSTS for short) [1, 13]. WSTSs are one example for infinite-state systems where (potentially) infinite sets of states can be represented (and effectively manipulated) symbolically in a finite way.

Our basic objects are data trees, i.e., trees with labels from an infinite domain. We view data trees as graphs, and define in a natural way a quasi-order on such graphs. Then we show that a uniform bound on the length of simple paths in such graphs, together with positive guards, makes DTPRS well-structured systems [1, 13]. As a technical tool we use here tree decompositions of graphs. In a nutshell we trade here recursion against positiveness, since considering both leads to undecidable static analysis. We show that for *positive-bounded* DTPRS, termination and tree pattern reachability are both decidable. On the negative side, we show that the verification of very simple Tree-LTL properties is undecidable even for *positive-bounded* DTPRS. On the positive side, the decidability result for pattern reachability can be extended to the verification of existential positive Tree-LTL properties. We then consider the type-checking problem, another static analysis problem, and show its Co-NexpTime-completeness for arbitrary DTPRS. Finally, we show that *bounded* model-checking of arbitrary DTPRS is NexpTime-complete.

Related work: Verification of web services often ignores unbounded data (c.f. e.g. [17, 14]). On the other hand, several data-driven workflow process models have been proposed. Document-driven workflow was proposed in [20]. Artifact-based workflow was outlined in [16], in which artifacts are used to represent key business entities, including both their data and life cycles. An early line of results involving data establishes decidability boundaries for the verification of temporal (first-order based) properties of a data-driven workflow processes, based on a relational data model [11, 10, 12]. This approach has been recently extended to the artifact-based model [9].

On the verification side, there is a rich literature on the verification of well-structured infinite transition systems [1, 13], ranging from faulty communication systems [7] to programs manipulating dynamic data [2] (citing only a few recent contributions). The latter work is one of the few examples where well-quasi-order on graphs is used.

Organization of the paper: In the next section, we fix some definitions and notations, then

define the DTPRS model. In Section 3 we show that analysis of DTPRS with recursive DTD or negated tree patterns is undecidable. In Section 4 we define positive-bounded DTPRS and prove our decidability results. Then in Section 5, we show the undecidability of the verification of simple Tree-LTL properties and the decidability of existential positive Tree-LTL properties for positive-bounded DTPRS. In Section 6, we consider the type-checking problem for DTPRS and show its Co-NexpTime completeness. Finally in Section 7, we consider bounded model-checking problem for DTPRS and show its NexpTime-completeness.

2 Definitions and notations

In this paper, documents are labeled, unranked, unordered trees. We fix a finite alphabet Σ (with symbols $a, b, c, \ldots$, called tags) and an infinite data domain $\mathcal{D}$ (with symbols $d, \ldots$). A *data tree* (see Figure 1) is a (rooted) tree T with nodes labeled by $\Sigma \cup \mathcal{D}$. A data tree T can be represented as a tuple $T = \langle V, E, \text{root}, \ell \rangle$, with labeling function $\ell : V \to \Sigma \cup \mathcal{D}$. Internal nodes are Σ-labeled, whereas leaves are $(\Sigma \cup \mathcal{D})$-labeled. We also fix a (finite) set of variables $\mathcal{X}$ (with symbols $X, Y, Z, \ldots$) that will take values in $\mathcal{D}$, and use $*$ as special symbol standing for any tag. Let $\mathcal{T}$ denote the set $\Sigma \cup \mathcal{X} \cup \{*\}$.

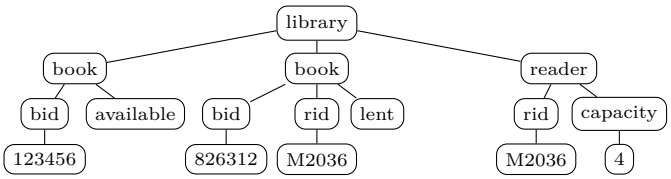


Figure 1 A document for a library system: The reader $M2036$ borrows a book with identifier 826312 from the library, and has capacity 4, namely he or she is able to borrow at most 4 other books.

We now introduce the different components used in our rewriting rules: *data tree patterns* to locate and specify a pattern of the document, *data constraints* to express data equalities and inequalities, *data tree pattern queries* to extract information from a document.

A *data constraint* is a Boolean combination of relations $X = Y$, with[1] $X, Y \in \mathcal{X}$. A *data tree pattern* (DTP) $P = \langle V, E, \text{root}, \ell, \tau, \mathit{cond} \rangle$ is a (rooted) $\mathcal{T}$-labeled tree $\langle V, E, \text{root}, \ell \rangle$, together with an edge-labeling function $\tau : E \to \{|, ||\}$ and a data constraint *cond*. Edges that are $|$-labeled denote child edges, and $||$-labeled edges denote descendant ones. Internal nodes are labeled by $\Sigma \cup \{*\}$, and leaves by $\mathcal{T}$. A *matching* of a DTP P into a data tree T is defined as a mapping preserving the root, the Σ- and $\mathcal{D}$-labels (with $*$ as wildcard), the child and the descendant relations, satisfying *cond* and mapping $\mathcal{X}$-labeled nodes to $\mathcal{D}$-labeled ones. In particular, a relation $X = Y$ $(X, Y \in \mathcal{X})$ in a DTP P means that the corresponding leaves of P must be mapped to leaves of T carrying the same data value. An *injective* matching of P into T means that the mapping above is injective. A *relative* DTP is a DTP with one designated node *self*. A relative DTP (P, self) is matched to a pair (T, v), where T is a tree and v is a node of T.

We use *Boolean combinations* of (relative) DTPs as rule guards. DTPs in a Boolean combination are matched *independently* of each other, except that nodes designated by *self*

[1] For simplicity we disallow here explicit data constants $X = d$ $(d \in \mathcal{D})$: they can be simulated by tags from Σ.

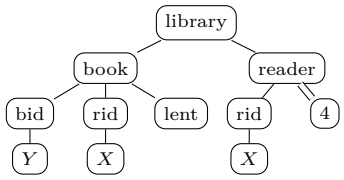


Figure 2 A data tree pattern (DTP).

must be matched to the same node of T. Boolean operators are interpreted by the standard meaning.

A *data tree pattern query* (DTPQ) is of the form $body \rightsquigarrow head$, with *body* a DTP and *head* a tree such that

- the internal nodes of *head* are labeled by Σ and its leaves are labeled by $(\Sigma \cup \mathcal{D} \cup \mathcal{X})$,
- every variable occurring in *head* also occurs in *body*,
- there is at least one variable occurring in *head*, i.e., at least one leaf of *head* is labeled by $\mathcal{X}$ (i.e., *head* is not a constant tree).

Let T be a data tree and $Q = body \rightsquigarrow head$ be a DTPQ. The evaluation result of Q over T is the forest $Q(T)$ of all instantiations of *head* by matchings from *body* to T. For example, the DTPQ $P \rightsquigarrow head$ with P given by Figure 2 and *head* consisting of a unique node labeled by Y, returns a forest consisting of one-node trees labeled by identifiers of books which are borrowed by readers with capacity 4. A *relative* DTPQ is defined like a DTPQ, except that its *body* is a relative DTP. A relative DTPQ Q is evaluated on a pair (T, v). The result of $Q(T, v)$ is defined as above, except that matchings of *body* must map node *self* to v. For instance, the relative DTPQ $P \rightsquigarrow head$ where the *reader* node is labeled *self* will return a forest of one-node trees labeled by identifiers of books which are borrowed by a particular reader with capacity 4 designated by *self*.

Similar to GAXML, data tree rewriting rules are guarded by (Boolean combinations of) DTPs and they can add information to a tree via queries. In addition, our rules can rename tags and delete nodes, together with their subtrees. Each rule comes along with a context called *locator*, that also describes all the operations related to a rewriting step. A locator L is a relative DTP with additional labels *append*, *del*, and ren_a $(a \in \Sigma)$. The meaning of these labels is to add information (*append*), delete a node and its subtree (*del*) and rename a tag into a (ren_a). Labels *append* and ren_a are not exclusive. They are restricted to be attached to nodes of L that are labeled by $\Sigma \cup \{*\}$. Label *del* can be attached to any node. We assume that all descendants in L of a node with *del* are also labeled by *del*, and that nodes labeled by *del* cannot be labeled by *append* or ren_a.

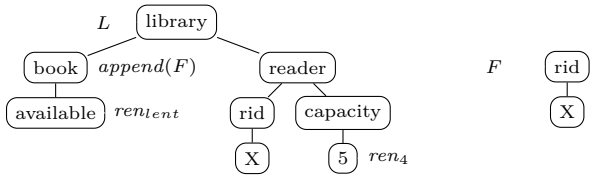


Figure 3 DTP rule "borrow": The reader identifier is added as a subtree to the node "book" whose state is renamed as "lent", and the capacity of the reader is decreased by renaming.

A *data tree pattern rewriting rule* (DTP rule for short) R is a tuple $\langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle$ with:

- L is a *locator*,

- G is a *guard*: a Boolean combination of (relative) DTPs,
- $\mathcal{Q}$ is a finite set of relative DTPQs,
- $\mathcal{F}$ is a finite set of forests with internal nodes labeled by Σ and leaves labeled by $\Sigma \cup \mathcal{D} \cup \mathcal{X} \cup \mathcal{Q}$,
- χ is a mapping from the set of nodes of L labeled by *append* to $\mathcal{F}$.

A *DTP rewriting system (DTPRS)* is a pair $(\mathcal{R}, \Delta)$ consisting of a finite set $\mathcal{R}$ of DTP rules and a *static invariant* Δ. The latter is a *DTD* τ together with a *data invariant* *inv*, i.e. a Boolean combination of DTPs. As usual for unordered trees, the DTD τ describes the syntax of trees. It is defined as a tuple $(\Sigma_r, \mathcal{P})$ such that Σ_r is the set of allowed root labels, and $\mathcal{P}$ is a finite set of rules $a \to \psi$ such that $a \in \Sigma$ and ψ is a Boolean combination of inequalities of the form $|b| \geq k$, where $b \in \Sigma \cup \{dom\}$ (*dom* is a symbol standing for any data value), and k is a non-negative integer. A *positive* DTD is one where the Boolean combinations above are positive. A *non-recursive* DTD is one where the rule graph is acyclic (the rule graph has Σ as vertex set and edges (a, b) for every a, b such that b occurs in a $a \to \psi$). For $\Delta = (\tau, inv)$ we write $T \models \Delta$ for a data tree T satisfying both τ and *inv*.

An example of a DTP rule for a reader of capacity 5 to borrow a book from a library is illustrated in Figure 3, including the locator L and $\mathcal{F} = \{F\}$.

We first describe the semantics of DTP rules informally. First, the locator is mapped against the data tree in a non-deterministic way. Then, queries are evaluated, thus determining the information that will possibly enhance the tree using the *append*-labels in the locator. Deletion and renaming are performed as expected. The resulting data tree must satisfy the static invariant Δ.

We now define the semantics formally. Let $T = \langle V, E, \text{root}, \ell \rangle$ be a data tree with $T \models \Delta$, and let $R = \langle L, G, \mathcal{Q}, \mathcal{F}, \chi \rangle$ be a DTP rule.

- Let μ be an *injective* matching from L to T. Let ν be the assignment of data values to variables in L such that $\nu(X) = \ell(\mu(v))$ for every v labeled by $X \in \mathcal{X}$ in L.
- For each variable $X \in \mathcal{X}$ we denote its evaluation as $X(T)$, with $X(T) = \nu(X)$ if defined, and $X(T)$ **a fresh data value otherwise**. Here a fresh data value is a data value which does not appear in T. Furthermore, it is required that all the *new* variables of R, i.e. variables occurring in $\mathcal{F}$, but not in L, should take **mutually distinct** fresh values. For each forest $F \in \mathcal{F}$, we denote its evaluation by $F(T)$, by replacing labels $Q \in \mathcal{Q}$ by $Q(T)$ and labels $X \in \mathcal{X}$ by $X(T)$. Recall that all queries $Q \in \mathcal{Q}$ are evaluated relatively to $\mu(self)$.
- A data tree T' is obtained from T by
  - deleting subtrees rooted at nodes $\mu(v)$ whenever v is labeled by *del* in L,
  - changing the tag of a node $\mu(v)$ to a whenever v is labeled by ren_a in L,
  - appending $F(T)$ as a subforest of nodes $\mu(v)$ whenever v is labeled by *append* in L and $\chi(v) = F$,
  - every other node of T keeps its tag or data.
- The rule R is enabled on data tree T if there exists an *injective* matching μ of L into T such that (1) the guard G is true on $(T, \mu(v))$ with v labeled by *self* in L, and (2) there is a data tree T', obtained from T and μ by the operations specified above, satisfying $T' \models \Delta$.

Let $T \xrightarrow{R} T'$ denote the transition from T to T' using DTP rule $R \in \mathcal{R}$.

▶ Remark. **1.** The injectivity of the matching μ ensures that the outcome of a rewriting step is well-defined. In particular, no two nodes with label *del* and *append* (or ren_a),

resp., can be mapped to the same node in the data tree. Notice that mappings used for guards or queries may be non-injective.

2. For the new variables occurring in $\mathcal{F}$, but not in L, we choose mutually distinct fresh values. We could have chosen arbitrary values instead, and enforce that they are fresh and mutually distinct *a posteriori* using the data invariant *inv*. In this case, *inv* needs negation. The *inv* (or the locator) can be also used to enforce that the (arbitrarily) chosen values already occur in T. This kind of invariant would be positive.
3. In our definition of DTP rules, it might appear that guards are redundant w.r.t. the locator. However, this is not the case in general, e.g. in the situation that guards include disjunctions or negations of DTPs.

Given a DTPRS $(\mathcal{R}, \Delta)$, let $T \longrightarrow T'$ denote the union of $T \xrightarrow{R} T'$ for some $R \in \mathcal{R}$, and $T \xrightarrow{+} T'$ (or $T \xrightarrow{*} T'$) denote the transitive (or reflexive and transitive) closure of $T \longrightarrow T'$. Moreover, let $\mathcal{T}^*_{\mathcal{R}}(T)$ denote the set of trees that can be reached from a data tree T by rewriting with DTP rules from $\mathcal{R}$, i.e. $\mathcal{T}^*_{\mathcal{R}}(T) = \{T' \mid T \xrightarrow{*} T'\}$. For a set of data trees $\mathcal{I}$, let $\mathcal{T}^*_{\mathcal{R}}(\mathcal{I})$ be the union of $\mathcal{T}^*_{\mathcal{R}}(T)$, for $T \in \mathcal{I}$.

We are interested in the following questions, given a DTPRS $(\mathcal{R}, \Delta)$:

- *Pattern reachability*: Given a DTP P and a set of initial trees[2] *Init*, given as the conjunction of a DTD and a Boolean combination of DTPs, is there some $T \in \mathcal{T}^*_{\mathcal{R}}(\mathit{Init})$ such that P matches T?
- *Termination:* Given an initial data tree T_0, are all runs (rewriting paths) $T_0 \to T_1 \to \cdots$ starting from T_0 finite?

The reason for the fact that termination of DTPRS is defined above w.r.t. a single initial data tree is that termination from a set of initial trees is already undecidable without data (see Proposition 3).

3 Undecidability

As one might expect, the analysis of DTPRS is quickly undecidable – and sometimes already without using any unbounded data. The proof of the proposition below is obtained by a straightforward simulation of 2-counter machines.

▶ Proposition 1. Both pattern reachability and termination for DTPRS $(\mathcal{R}, \Delta)$ are undecidable whenever one of the following holds:

1. the DTD in Δ is recursive,
2. either guards in $\mathcal{R}$ or the invariant Δ contain negated DTPs.

The above result holds already without data.

The next result shows that with data, we can relax both conditions above and still get undecidability of DTPRS. The main idea is to use data for creating long horizontal paths (although trees are supposed to be unordered). Such horizontal paths can be obtained e.g. with a tree of depth 2, with each subtree (of the root) containing three nodes, a node plus its two children labeled respectively by data values d_i, d_{i+1}. Assuming all data values d_i are distinct (and distinguishing d_1), then a linear order on these subtrees is obtained.

▶ **Theorem 2.** *Both pattern reachability and termination are undecidable for DTPRS $(\mathcal{R}, \Delta)$ such that (1) the DTD in Δ is non-recursive and (2) all DTPs from guards in $\mathcal{R}$ and the invariant Δ are positive.*

[2] We require that every tree in *Init* satisfies Δ.

We end this section with a remark on the undecidability of termination from an *initial set of trees*. First we notice that – already without data – DTPRS can simulate so-called *reset Petri nets* [15]. These are Petri nets (or equivalently, multi-counter automata without zero test) with additional transitions that can reset places (equivalently, counters) to zero. They can be represented by trees of depth 2, where nodes at depth one represent places, and their respective number of children (leaves) is the number of tokens on that place. A DTPRS (without data) can easily simulate increments, decrements and resets (using deletion in DTPRS). It is known that so-called *structural termination* for reset Petri nets is undecidable [18], i.e., the question whether there are infinite computations from *any* initial configuration, is undecidable. This implies:

▶ Proposition 3. The following question is undecidable: Given a DTPRS $(\mathcal{R}, \Delta)$, is there some tree T_0 satisfying Δ and an infinite computation $T_0 \longrightarrow T_1 \longrightarrow \cdots$ in $(\mathcal{R}, \Delta)$? This holds already for non-recursive DTD in Δ and without data constraints in DTPs.

It follows from Proposition 3 that termination from an initial set of trees, namely to decide whether for every $T_0 \in \mathit{Init}$, all the runs starting from T_0 terminate, is undecidable.

4 Positive-bounded DTPRS

In this section we consider *positive-bounded DTPRS*, a fragment of DTPRS for which we show that pattern reachability and termination are decidable.

From Proposition 1, we know that in order to get decidability, the DTD in the static invariant Δ must be non-recursive. For a non-recursive DTD, there is some B such that every tree satisfying the DTD has depth bounded by B. In the following, we assume the existence of such a bound B. Also from Proposition 1, we know that for obtaining decidability we need to restrict ourselves to positive guards and positive data invariants.

However, from Theorem 2, we know that these restrictions alone do not suffice to achieve decidability. We also need to disallow long linear orders created by data. For this, we introduce a last restriction, called *simple-path bounded*, which is defined in the following.

Let $T = \langle V, E, root, \ell \rangle$ be a data tree. *The graph $G(T)$ associated with T* is the undirected graph obtained from T by merging all the nodes labeled by the same $d \in \mathcal{D}$ into a single node d. Formally, $G(T) = (V', E')$, where $V' = \{v \in V \mid \ell(v) \in \Sigma\} \cup \{\ell(v) \mid v \in V, \ell(v) \in \mathcal{D}\}$ and $E' = \{\{v, w\} \mid \ell(v), \ell(w) \in \Sigma, (v, w) \in E\} \cup \{\{v, d\} \mid \ell(v) \in \Sigma, \exists w \text{ s.t. } (v, w) \in E, \ell(w) = d\}$. A *simple path* of T is a simple path in $G(T)$, i.e. a sequence of vertices $v_1, \ldots, v_n$ in $G(T)$ such that for all $i \neq j$, $\{v_i, v_{i+1}\} \in E'$ and $v_i \neq v_j$. The length of a path $v_1, \ldots, v_n$ is $n - 1$.

Formally, a DTPRS $(\mathcal{R}, \Delta)$ is *positive-bounded* with set of initial trees *Init*, if:

- **non-recursive-DTD:** the DTD in the static invariant Δ is non-recursive. In particular, trees satisfying the DTD have depth bounded by some $B > 0$.
- **positive:** all guards in $\mathcal{R}$ and the data invariant in Δ are positive Boolean combinations of DTPs. The DTD in Δ is positive as well.
- **simple-path bounded:** there exists $K > 0$ such that for any $T_0 \in \mathit{Init}$, the length of any simple path in any $T \in \mathcal{T}^*_{\mathcal{R}}(T_0)$, is bounded by K.

Notice that the third condition above implies that all data trees have depth bounded by K. So we always assume that $B \leq K$. Notice also that in positive-bounded DTPRS, the data value inequality is *allowed* in DTPs, that is, we can state that two data values are different. Moreover, fresh data values (for the new variables in DTP rules) can be used to model some conceptually negative data constraints, like for instance key properties. Notice also that there is no restriction on the DTD of the initial set *Init*. However, since the DTD in the invariant Δ is positive, "at most"-constraints must be ensured via the rewriting rules.

The library example illustrated in Figure 1 includes DTP rules for book borrowing (Figure 3) and returning, the registration of new books and new readers (where fresh data values can be used to guarantee that the *rid* and *bid* are "keys"), and the deletion of reader accounts. It is easy to notice that the library example satisfies the first 2 conditions above. It is also the case for the third condition. Indeed, all simple paths are bounded by 7: A longest path is for instance: library - book - rid - $M2036$ - rid - reader - capacity - 4. Notice that the bound still holds even if the capacity of a reader is unbounded.

The rest of the section is devoted to the proof of the following result:

▶ **Theorem 4.** *Given a positive-bounded DTPRS $(\mathcal{R}, \Delta)$, pattern reachability and termination are both decidable.*

We prove Theorem 4 by using the framework of *well-structured transition systems (WSTS)* [1, 13], which has been applied to DTPRS *without* data in [15]. We recall briefly some definitions. A WSTS is a triple $(S, \longrightarrow, \preceq)$ such that S is an (infinite) state space, $\preceq$ is a *well-quasi-ordering*[3] *(wqo for short)* on S, and $\longrightarrow$ is the transition relation on S. It is required that $\longrightarrow$ is *compatible w.r.t.* $\preceq$: for any $s, t, s' \in S$ with $s \longrightarrow t$ and $s \preceq s'$, there exists $t' \in S$ such that $s' \longrightarrow t'$ and $t \preceq t'$.

Let $\mathcal{T}_{B,K}$ denote the set of data trees whose depths are bounded by B and lengths of simple paths are bounded by K. From the definition of positive-bounded DTPRS, we know that $\mathcal{T}^*_{\mathcal{R}}(\mathit{Init}) \subseteq \mathcal{T}_{B,K}$. In the following, we prove Theorem 4 by defining a binary relation $\preceq$ on $\mathcal{T}_{B,K}$ and showing that $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ is a WSTS.

4.1 Well-structure of positive-bounded DTPRS

We define a binary relation $\preceq$ on $\mathcal{T}_{B,K}$ as follows. Let $T_1 = \langle V_1, E_1, \mathrm{root}_1, \ell_1\rangle, T_2 = \langle V_2, E_2, \mathrm{root}_2, \ell_2\rangle \in \mathcal{T}_{B,K}$, then $T_1 \preceq T_2$ if there is an injective mapping ϕ from V_1 to V_2 such that

- root preservation: $\phi(\mathrm{root}_1) = \mathrm{root}_2$,
- parent-child relation preservation: $(v_1, v_2) \in E_1$ iff $(\phi(v_1), \phi(v_2)) \in E_2$,
- tag preservation: If $\ell_1(v) \in \Sigma$, then $\ell_1(v) = \ell_2(\phi(v))$,
- data value (in)equality preservation: If $v_1, v_2 \in V_1$ and $\ell_1(v_1), \ell_1(v_2) \in \mathcal{D}$, then $\ell_2(\phi(v_1)), \ell_2(\phi(v_2)) \in \mathcal{D}$, and $\ell_1(v_1) = \ell_1(v_2)$ iff $\ell_2(\phi(v_1)) = \ell_2(\phi(v_2))$.

It is easy to see that $\preceq$ is reflexive and transitive, so it is a quasi-order. In the following, we first assume that $\preceq$ is a wqo on $\mathcal{T}_{B,K}$ and show that $\longrightarrow$ is compatible with $\preceq$, in order to prove Theorem 4. We show in Section 4.2 that $\preceq$ is indeed a wqo: for any infinite sequence of data trees $T_0, T_1, \ldots \in \mathcal{T}_{B,K}$, there are $i < j$ such that $T_i \preceq T_j$.

▶ Proposition 5. Let $(\mathcal{R}, \Delta)$ be a positive-bounded DTPRS. Let $T_1, T_1', T_2 \in \mathcal{T}_{B,K}$, $T_1 \xrightarrow{R} T_2$ for some $R \in \mathcal{R}$, and $T_1 \preceq T_1'$. Then there exists $T_2' \in \mathcal{T}_{B,K}$ such that $T_1' \xrightarrow{R} T_2'$ and $T_2 \preceq T_2'$.

Consequently, in the positive-bounded fragment $\longrightarrow$ is compatible w.r.t. $\preceq$ in $\mathcal{T}_{B,K}$, thus $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ is a WSTS.

In addition, it can be shown that $(\mathcal{T}_{B,K}, \longrightarrow, \preceq)$ satisfies some additional computability conditions which are needed to show the decidability of pattern reachability and termination, namely, *effectiveness of pred-basis* for pattern reachability and *effectiveness of successor relation* for termination. With these computability conditions, Theorem 4 then follows from the properties of WSTS (c.f. Theorem 3.6 and Theorem 4.6 in [13]).

[3] A wqo $\preceq$ is a reflexive, transitive and well-founded relation with no infinite antichain.

4.2 Well-quasi-ordering for data trees

In order to prove that $\preceq$ is a wqo over $\mathcal{T}_{B,K}$, we first represent a data tree T as a (labeled) undirected graph $G_\ell(T)$, then we encode $G_\ell(T)$ into a tree (without data) of *bounded depth* using the concept of tree decompositions. Define a binary relation $\leq$ on labeled trees (without data) of bounded depth as follows: $T_1 \leq T_2$ if there is an injective mapping from T_1 to T_2 preserving the root, the tags, and the parent-child relation. It is known that $\leq$ is a wqo on labeled trees of bounded depth *without data* [15].

Let $\mathcal{G}_K$ be the set of labeled graphs with the lengths of all simple paths bounded by K. In the following, we show that $\preceq$ on $\mathcal{T}_{B,K}$ corresponds to the induced subgraph relation (formally defined later) on $\mathcal{G}_K$, and the fact that $\leq$ is a wqo for labeled trees of bounded depth implies that the induced subgraph relation is a wqo on $\mathcal{G}_K$.

Given a data tree $T = \langle V, E, root, \ell\rangle \in \mathcal{T}_{B,K}$, *the labeled undirected graph representation* $G_\ell(T)$ *of* T is obtained from $G(T)$, the graph associated to T, by adding labels encoding information of data tree nodes (tag, depth ...). Formally, $G_\ell(T)$, is a $((\Sigma \times [B+1]) \cup \{\$\})$-labeled (where $[B+1] = \{0, 1, \cdots, B\}$) undirected graph (V', E', ℓ') defined as follows,

- $V' = \{v \in V \mid \ell(v) \in \Sigma\} \cup \{\ell(v) \mid v \in V, \ell(v) \in \mathcal{D}\}$,
- $E' = \{\{v, w\} \mid \ell(v), \ell(w) \in \Sigma, (v, w) \in E\} \cup \{\{v, d\} \mid \ell(v) \in \Sigma, \exists w, (v, w) \in E, \ell(w) = d\}$,
- Let $v \in V$ such that $\ell(v) \in \Sigma$, then $\ell'(v) = (\ell(v), i)$. In addition, $\ell'(d) = \$$ for each $d \in V' \cap \mathcal{D}$.

Let Σ_G denote $(\Sigma \times [B+1]) \cup \{\$\}$. For Σ_G-labeled graphs, we define the induced subgraph relation as follows. Let $G_1 = (V_1, E_1, \ell_1), G_2 = (V_2, E_2, \ell_2)$ be two Σ_G-labeled graphs, then G_1 is an *induced subgraph* of G_2 (denoted $G_1 \sqsubseteq G_2$) iff there is an injective mapping ϕ from V_1 to V_2 such that

- label preservation: $\ell_1(v_1) = \ell_2(\phi(v_1))$ for any $v_1 \in V_1$,
- edge preservation: let $v_1, v_1' \in V_1$, then $\{v_1, v_1'\} \in E_1$ iff $\{\phi(v_1), \phi(v_1')\} \in E_2$.

From the definition of the labeled graph representation of data trees, it is not hard to show that the induced subgraph relation $\sqsubseteq$ corresponds to the relation $\preceq$ on data trees.

▶ Proposition 6. Let $T_1, T_2 \in \mathcal{T}_{B,K}$, then $T_1 \preceq T_2$ iff $G_\ell(T_1) \sqsubseteq G_\ell(T_2)$.

Now we show how to encode any Σ_G-labeled graph belonging to $\mathcal{G}_K$ into a labeled tree of bounded depth by using tree decompositions.

Let $G = (V, E, \ell)$ be a connected Σ_G-labeled graph, then a *tree decomposition* of G is a quadruple $T = \langle U, F, r, \theta\rangle$ such that:

- (U, F, r) is a tree with the domain U, the parent-child relation F, and the root $r \in U$,
- $\theta : U \to 2^V$ is a labeling function attaching each node $u \in U$ a set of vertices of G,
- For each edge $\{v, w\} \in E$, there is a node $u \in U$ such that $\{v, w\} \subseteq \theta(u)$,
- For each vertex $v \in V$, the set of nodes $u \in U$ such that $v \in \theta(u)$ constitutes a connected subgraph of T.

The sets $\theta(v)$ are called the *bags* of the tree decomposition. The *depth* of a tree decomposition $T = \langle U, F, r, \theta\rangle$ is the depth of the tree (U, F, r) and the *width* of T is defined as $\max\{|\theta(u)| - 1 \mid u \in U\}$. The *tree-width* of a graph $G = (V, E)$ is the minimum width of tree decompositions of G. For a tree decomposition of width K of a graph G, without loss of generality, we assume that each bag is given by a sequence of vertices of length $K + 1$, $v_0 \dots v_K$, with possible repetitions, i.e. possibly $v_i = v_j$ for some $i \neq j$ (tree decompositions in this form are sometimes called ordered tree decompositions).

▶ **Theorem 7.** *([19, 6]) If $G \in \mathcal{G}_K$, then G has a tree decomposition with both depth and width bounded by K.*

Now we describe how to encode labeled graphs by trees using tree decompositions.

Let $G = (V, E, \ell) \in \mathcal{G}_K$ be a Σ_G-labeled graph, and $T = \langle U, F, r, \theta \rangle$ be a tree decomposition of G with width K and depth at most K. Remember that each $\theta(u)$ is represented as a sequence of exactly $K+1$ vertices, and $[K+1] = \{0, \ldots, K\}$. Define

$$\Sigma_{G,K} := (\Sigma_G)^{K+1} \times 2^{[K+1]^2} \times 2^{[K+1]^2} \times 2^{[K+1]^2}.$$

We transform $T = \langle U, F, r, \theta \rangle$ into a $\Sigma_{G,K}$-labeled tree $T' = (U, F, r, \eta)$, which encodes in a uniform way the information about G (including edge relations and vertex labels). $\eta : U \to \Sigma_{G,K}$ is defined as follows. Let $\theta(u) = v_0 \ldots v_K$, then $\eta(u) = (\ell(v_0) \ldots \ell(v_K), \overline{\lambda})$, where $\overline{\lambda} = (\lambda_1, \lambda_2, \lambda_3)$,

- $\lambda_1 = \{(i,j) \mid 0 \leq i, j \leq K, v_i = v_j\}$,
- $\lambda_2 = \{(i,j) \mid 0 \leq i, j \leq K, \{v_i, v_j\} \in E\}$,
- If $u = r$, then $\lambda_3 = \emptyset$, otherwise let u' be the parent of u in T and $\theta(u') = v'_0 \cdots v'_K$, then $\lambda_3 = \{(i,j) \mid 0 \leq i, j \leq K, v'_i = v_j\}$.

The encoding of labeled graphs into labeled trees establishes a connection between the wqo $\leq$ of labeled trees and the induced subgraph relation ($\sqsubseteq$) of labeled graphs.

▶ Proposition 8. Let G_1, G_2 be two Σ_G-labeled graphs with tree-width bounded by K, and T_1, T_2 be two tree decompositions of width K of resp. G_1, G_2, then the two $\Sigma_{G,K}$-labeled trees T'_1, T'_2 obtained from T_1, T_2 satisfy that: If $T'_1 \leq T'_2$, then $G_1 \sqsubseteq G_2$.

Now we are ready to show that $\preceq$ is a wqo for $\mathcal{T}_{B,K}$. Let $T_0, T_1, \ldots$ be an infinite sequence of data trees from $\mathcal{T}_{B,K}$. Consider the infinite sequence of $\Sigma_{G,K}$-labeled trees $T'_0, T'_1, \ldots$ obtained from the tree decompositions (with width K and depth at most K) of graphs $G_\ell(T_0), G_\ell(T_1), \ldots$. Then there are $i, j : i < j$ such that $T'_i \leq T'_j$, because $\leq$ is a wqo for labeled trees of depth at most K. So $G_\ell(T_i) \sqsubseteq G_\ell(T_j)$ from Proposition 8, and $T_i \preceq T_j$ from Proposition 6. We thus prove following theorem.

▶ **Theorem 9.** *$\preceq$ is a well-quasi-ordering over $\mathcal{T}_{B,K}$.*

5 Verification of temporal properties

Until now we considered only two properties for static analysis: pattern reachability and termination. (Non-)reachability of a DTP can be expressed easily in Tree-LTL [5], which corresponds roughly to linear time temporal logics where atomic propositions are DTPs[4]. We show in this section that allowing for runs of unbounded length makes the validation of (even very simple) Tree-LTL properties undecidable, even without data:

▶ **Theorem 10.** *It is undecidable whether a positive-bounded DTPRS satisfies a Tree-LTL formula $F\varphi$, where φ is a positive Boolean combination of DTPs. This holds already without data.*

The proof of Theorem 10 is by a reduction from the halting problem of two-counter machines. The idea is to simulate a two-counter machine by a positive bounded DTPRS ignoring the zero-tests, and describe them by a Tree-LTL formula $F\varphi$. The proof relies on the universal semantics of Tree-LTL, requiring that every run of the DTPRS satisfies the formula.

[4] Such formulas use actually free variables in patterns, which are then quantified universally. This is consistent with the approach of testing whether a model satisfies the negation of a formula.

If the *existential* semantics of Tree-LTL formulas is used instead, i.e. requiring that there is a run of the DTPRS satisfying a given Tree-LTL formula, then the problem is still undecidable if *negations* are available, since the negation of $F\varphi$ in the universal semantics is $G\neg\varphi$ in the existential semantics. If negations are also forbidden, then we get decidability:

▶ Proposition 11. It is decidable whether a positive-bounded DTPRS satisfies a given positive existential Tree-LTL formula defined by the following rules,

$$\varphi ::= true \mid false \mid P \mid \varphi_1 \vee \varphi_2 \mid \varphi_1 \wedge \varphi_2 \mid X\varphi_1 \mid \varphi_1 U \varphi_2,$$

where P is a DTP.

6 Type-checking DTPRS

This section shows that it can be checked statically whether DTP rules preserve the static invariant $\Delta = (\tau, inv)$ consisting of a DTD τ and a data invariant inv.

Recall that in the definition of DTPRS, if $T \xrightarrow{R} T'$, then it is required that $T' \models \Delta$. Here we drop this requirement and consider the following type-checking problem.

DTPRS Type-checking: Given a DTPRS $(\mathcal{R}, \Delta)$ with a non-recursive DTD[5], decide whether for each T, T' and each DTP rule R such that $T \models \Delta$ and $T \xrightarrow{R} T'$, it holds that $T' \models \Delta$.

▶ **Theorem 12.** *DTPRS type-checking is Co-NexpTime-complete.*

The upper bound of Theorem 12 is shown by a small model argument. The lower bound follows from [8], that shows that satisfiability of DTPs on depth-bounded data trees relative to a DTD is NexpTime-hard.

7 Bounded model-checking DTPRS

In this section we consider *bounded model-checking* for DTPRS: Given a DTPRS $(\mathcal{R}, \Delta)$ with a non-recursive DTD [6], a set of initial trees *Init*, a DTP P and a bound N (encoded in unary) we ask whether there is some T_0 satisfying *Init* and some T s.t. P matches T and $T_0 \xrightarrow{\leq N} T$. We have the following result:

▶ **Theorem 13.** *Bounded model-checking for DTPRS is NexpTime-complete.*

Theorem 13 can be extended to bounded model-checking Tree-LTL properties. Bounded model-checking of a Tree-LTL formula φ with a bound N is the problem checking whether a counter-example for φ can be obtained in at most N rewriting steps. For instance, bounded model-checking for $G\neg P$ with a bound N is to check whether the DTP P can be reached in $\leq N$ steps.

The proof of Theorem 13 follows the similar line as the proof of Co-2NexpTime completeness of model-checking Tree-LTL properties over recursion-free GAXML ([5]): The upper-bound is shown by a (exponential) small-model property, and the lower-bound is shown by a simulation of the computations of NexpTime-Turing machines. The gap between

[5] If the DTD is recursive, then the problem is undecidable, since the satisfiability of Boolean combinations of DTPs over a recursive DTD is undecidable [8].

[6] The recursive DTD will quickly lead to undecidability, as argued for the type-checking problem.

the NexpTime complexity above and the Co-2NexpTime complexity in [5] is essentially due to the unary encoding of the bound N for bounded model checking.

Acknowledgements We would like to thank the participants of the ANR Docflow and the CREATE Activedoc projects for the discussion on the verification of AXML systems, and in particular Serge Abiteboul and Albert Benveniste.

References

1 P. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite-state systems. In *LICS'96*, pages 313–321. IEEE, 1996.
2 P. A. Abdulla, A. Bouajjani, J. Cederberg, F. Haziza, and A. Rezine. Monotonic abstraction for programs with dynamic memory heaps. In *CAV'08*, volume 5123 of *LNCS*, pages 341–354. Springer, 2008.
3 S. Abiteboul, O. Benjelloun, and T. Milo. Positive Active XML. In *PODS'04*, pages 35–45. ACM, 2004.
4 S. Abiteboul, O. Benjelloun, and T. Milo. The Active XML project: an overview. *VLDB Journal*, 17(5):1019–1040, 2008.
5 S. Abiteboul, L. Segoufin, and V. Vianu. Static analysis of active XML systems. In *PODS'08*, pages 221–230. ACM, 2008. Journal version in *ACM Trans. Database Syst.* 34(4): 2009.
6 A. Blumensath and B. Courcelle. On the monadic second-order transduction hierarchy. HAL Archive, 2009. http://hal.archives-ouvertes.fr/hal-00287223/fr.
7 P. Bouyer, N. Markey, J. Ouaknine, P. Schnoebelen, and J. Worrell. On termination for faulty channel machines. In *STACS'08*, pages 121–132, 2008.
8 C. David. Complexity of data tree patterns over XML documents. In *MFCS '08*, pages 278–289, 2008.
9 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT'09*, pages 252–267, 2009.
10 A. Deutsch, L. Sui, and V. Vianu. Specification and verification of data-driven web applications. *JCSS*, 73(3):442–474, 2007.
11 A. Deutsch, L. Sui, V. Vianu, and D. Zhou. Verification of communicating data-driven web services. In *PODS'06*, pages 90–99, 2006.
12 A. Deutsch and V. Vianu. WAVE: Automatic verification of data-driven web services. *IEEE Data Eng. Bull.*, 31(3):35–39, 2008.
13 A. Finkel and P. Schnoebelen. Well-structured transition systems everywhere! *TCS*, 256(1-2):63–92, 2001.
14 X. Fu, T. Bultan, and J. Su. Conversation protocols: a formalism for specification and verification of reactive electronic services. *TCS*, 328(1-2):19–37, 2004.
15 B. Genest, A. Muscholl, O. Serre, and M. Zeitoun. Tree pattern rewriting systems. In *ATVA'08*, pages 332–346. Springer, 2008.
16 R. Hull. Artifact-centric business process models: Brief survey of research results and challenges. In *OTM'08*, pages 1152–1163, 2008.
17 R. Hull, M. Benedikt, V. Christophides, and S. Jianwen. E-services: a look behind the curtain. In *PODS'03*, pages 1–14, 2003.
18 R. Mayr. Undecidable problems in unreliable computations. *TCS*, 297(1-3):337–354, 2003.
19 J. Nešetřil and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *Eur. J. Comb.*, 27(6):1022–1041, 2006.
20 J. Wang and A. Kumar. A framework for document-driven workflow systems. In *BPM'05*, pages 285–301, 2005.

Temporal Logics on Words with Multiple Data Values*

Ahmet Kara[1], Thomas Schwentick[1], and Thomas Zeume[1]

1 **TU Dortmund**
Germany
{ahmet.kara, thomas.schwentick, thomas.zeume}@cs.tu-dortmund.de

Abstract

The paper proposes and studies temporal logics for attributed words, that is, data words with a (finite) set of (attribute,value)-pairs at each position. It considers a basic logic which is a semantical fragment of the logic $\mathrm{LTL}_1^{\downarrow}$ of Demri and Lazic with operators for navigation into the future and the past. By reduction to the emptiness problem for data automata it is shown that this basic logic is decidable. Whereas the basic logic only allows navigation to positions where a fixed data value occurs, extensions are studied that also allow navigation to positions with different data values. Besides some undecidable results it is shown that the extension by a certain UNTIL-operator with an inequality target condition remains decidable.

1998 ACM Subject Classification F.4.1 [Mathematical Logic and Formal Languages]: Mathematical Logic – Temporal logic; F.4.3 [Mathematical Logic and Formal Languages]: Formal Languages – Decision problems

Keywords and phrases Expressiveness, Decidability, Data words

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.481

1 Introduction

Motivated by questions from XML theory and automated verification, extensions of (finite or infinite) strings by data values from unbounded domains have been studied intensely in recent years. Various logics and automata for such data words have been invented and investigated.

A very early study by Kaminski and Francez [16] considered automata on strings over an "infinite alphabet". In [7], *data words* were invented as finite sequences of pairs (σ, d), where σ is a symbol from a finite alphabet and d a value from a possibly infinite domain. In [6] multi-dimensional data words were considered where every position carries N variable valuations, for some fixed N. Similar models can be found for instance in [2] and other work on parameterized verification. More powerful models were investigated in [19] and [14] where every position is labeled by the state of a relational database, i.e., by a set of relations over a fixed signature.

For the basic model of data strings with one data value per position a couple of automata models and logics have been invented and their algorithmic and expressive properties have been studied. On the automata side we mention register automata [16, 8, 22] (named *finite memory automata* in [16, 8]), pebble automata [22, 24], alternating 1-register automata [12], data automata [5] (or the equivalent class memory automata [3]).

* We acknowledge the financial support by the German DFG under grant SCHW 678/4-1.

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 481–492

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

On the logical side, classical logics like two-variable first-order logic [5] have been studied and recently order comparisons between data values have been considered [20, 23]. The satisfiability problem for two-variable first-order logic over data words is decidable if data values can only be compared for equality but positions can be compared with respect to the linear order and the successor relation [5]. However, the complexity is unknown. It is elementary if and only if testing reachability in Petri nets is elementary as well [5]. The proof of decidability uses *data automata*, a strong automata model with decidable non-emptiness.

More relevant for this paper are previous investigations of temporal logics on data words. A pioneering contribution was by Demri and Lazic [12] (the journal version of [11]) which introduced Freeze LTL. In a nutshell, Freeze LTL extends LTL by *freeze quantifiers* which[1] allow to "store" the current data value in a register and to test at a possibly different position whether that position carries the same value. Freeze LTL has a decidable finite satisfiability problem if it is restricted to one register ($\mathrm{LTL}_1^{\downarrow}$) and to future navigation, but the complexity is not primitive recursive. With one register and past (and future) navigation it is undecidable. In [15] it is shown that these lower bounds even hold if only navigation with F and P (but without X) are allowed.

In [12], also a restriction of $\mathrm{LTL}_1^{\downarrow}$, *simple* $LTL_1^{\downarrow}$, was investigated and it was shown that it is expressively equivalent to two-variable logics. The restriction requires that (syntactically) between each value test and the corresponding freeze quantifier there is at most one temporal operator and it disallows Until and Since navigation but allows past navigation. Thanks to the (effective) equivalence to two-variable logics, simple $\mathrm{LTL}_1^{\downarrow}$ is decidable.

One of our aims in this paper was to find a decidable temporal logic on data words with past navigation that is more expressive than simple $\mathrm{LTL}_1^{\downarrow}$. In particular it should allow Until navigation with reference to data values. On the other hand, the logics we study are semantical fragments of $\mathrm{LTL}_1^{\downarrow}$. Furthermore this work was motivated by the decidable logic $\mathrm{CLTL}^{\diamond}$ for multi-attribute data words [10]. It allows to test whether somewhere in the future (or past) a current data value occurs and it can compare data values between two positions of bounded distance. The logics proposed in this paper are intended to have more expressive power than $\mathrm{CLTL}^{\diamond}$ while retaining its decidability.

Contribution

We propose and investigate temporal logics for multi-attribute data words. An attributed word is a string which can have a finite number of (attribute,value)-pairs at each position (in the spirit of XML) and has propositions rather than symbols (in the spirit of LTL).

We first define Basic Data LTL which mimics the navigation abilities of simple $\mathrm{LTL}_1^{\downarrow}$, if only positive register tests are used. As sequences of such navigation steps do not do any harm we drop the requirement to freeze the data value at every step and replace freeze quantifiers by a class quantifier which restricts a sub-formula to the positions at which this data value appears. We show that a slight extension of this logic captures simple $\mathrm{LTL}_1^{\downarrow}$ (Proposition 2) and that it is decidable (Theorem 1). Although strictly more expressive than $\mathrm{CLTL}^{\diamond}$, the decidability proof for Basic Data LTL is conceptually simpler than the proof given in [10]. It uses an encoding of multi-attribute words by data words and a reduction to non-emptiness of data automata. A similar multi-attribute encoding has already been used in [13]. The result generalizes to attributed ω-words (Theorem 3). Some obvious extensions (by navigation with respect to two data values or Until navigation where intermediate positions

[1] We note that the freeze quantifier itself was used already in [9] and in previous work, e.g., in [1].

can be tested by data-free formulas) are undecidable (Theorems 4 and 6, respectively).

Finally, we add a powerful Until-operator to Basic Data LTL, which allows to navigate to a position with a data value that is *different* from the value of a given attribute at the starting position. Furthermore, it can test properties of intermediate positions by arbitrary sub-formulas and can even test (in a limited way) whether intermediate positions have attribute values different from or equal to the value on the starting position. The resulting logic can express all properties expressible in two-variable first-order logic and contains the Until operator. That this logic is still decidable is the main technical contribution of the paper.

The paper is organized as follows. In Section 2, we define attributed words and Basic Data LTL and give some example properties. In Section 3, we compare Basic Data LTL with other logics. Section 4 shows that Basic Data LTL is decidable and presents undecidability results for some extensions. Section 5 introduces the extended Until operator and shows decidability of the resulting logic. It also shows (the simple fact) that an Until-operator that navigates with respect to equality and allows (only) data-free intermediate tests quickly leads to an undecidable logic. We conclude in Section 6. Due to lack of space most proofs are only sketched or even missing. They can be found in the full version of the paper [17].

Related work

We discussed many related papers above. Another approach, combining temporal and classical logics, was studied in [14]. It allows to navigate by temporal operators and to evaluate first-order formulas in states. Properties depending on values at different states can be stated by global universal quantification of values. In [6] a first-order logic on multi-dimensional data words was studied.

Acknowledgements

The idea to extend the temporal logic that is equivalent to two-variable logics by Until operators (without reference to data) goes back to a suggestion by Mikołaj Bojanczyk [4]. We are also indebted to Volker Weber with whom we carried out first investigations before he tragically passed away in 2009. The remarks by the reviewers of FSTTCS 2010 helped to improve the presentation and to add some additional references.

2 Definitions

We first fix the data model and define BD-LTL afterwards. Finally we give an example that illustrates the way in which properties can be expressed

2.1 Attributed words

Let $\mathcal{PROP}$ and $\mathcal{ATT}$ be (possibly infinite) sets of propositions and attributes and $\mathcal{D}$ an infinite set of data values. An *attributed word* w is a finite word where every position carries a finite set $\{p_1, \ldots, p_l\}$ of propositions from $\mathcal{PROP}$ and a finite set $\{(a_1, d_1), \ldots, (a_k, d_k) \mid a_i \neq a_j \text{ for } i \neq j\}$ of attribute-value pairs from $\mathcal{ATT} \times \mathcal{D}$.

Given an attributed word w we denote the proposition set of position i in w by $w[i].\mathcal{P}$. A position i is a *p-position* if $p \in w[i].\mathcal{P}$. By $w[i].@a$ we denote the value of attribute a on position i. If position i does not carry attribute a, then $w[i].@a = nil \notin D$. The *word projection* of an attributed word $w = w_1 \ldots w_n$ is defined by $str(w) := w[1].\mathcal{P} \ldots w[n].\mathcal{P}$. By $pos_d(w)$ we denote the set of *class positions* of d in w, that is, the set of positions of w with

at least one attribute with value d. The *class word* $class_d(w)$ of w with respect to d is the restriction of w to the positions of $pos_d(w)$.

We always consider sets of words over some finite set $\mathcal{P}$ of propositions and a finite set $\mathcal{V}$ of attributes[2]. We call an attributed word w $\mathcal{V}$*-complete* for a finite set $\mathcal{V} \subseteq \mathcal{ATT}$ if every position of w has exactly one pair (a, d_a) for each $a \in \mathcal{V}$. A $\{a\}$-complete word is called *1-attributed word* . We refer to the value of attribute $@a$ at a position i in a 1-attributed word as *the data value of* i. There is an immediate correspondence between data strings (that is, sequences of (symbol,value) pairs) and 1-attributed words. Thus, we use in this paper automata and logics that were introduced for data strings also for 1-attributed words.

Attributed ω-words are defined accordingly.

For $i, j \in \mathbb{N}$ with $i \leq j$ we denote the interval $\{i, i+1, \dots j\}$ by $[i, j]$. As usual we use round brackets to denote open intervals, e.g., $[3, 5) = \{3, 4\}$.

2.2 Basic Data LTL

The logic Basic Data LTL (abbreviated: BD-LTL) has two main types of formulas, *position formulas* and *class formulas*, where, intuitively, class formulas express properties of class words. We first state the syntax of the logic and give an intuitive explanation of its non-standard features afterwards.

We fix a finite set $\mathcal{P} \subseteq \mathcal{PROP}$ of propositions and a finite set $\mathcal{V} \subseteq \mathcal{ATT}$ of attributes.

The syntax of *position formulas* φ and *class formulas* ψ of BD-LTL (over $\mathcal{P}$ and $\mathcal{V}$) are defined as follows.

$$\begin{aligned} \varphi &::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \mathrm{X}\varphi \mid \mathrm{Y}\varphi \mid \varphi \mathrm{U} \varphi \mid \varphi \mathrm{S} \varphi \mid \mathrm{C}^{\delta}_{@a}\psi \\ \psi &::= \varphi \mid @a \mid \neg\psi \mid \psi \vee \psi \mid \mathrm{X}^{=}\psi \mid \mathrm{Y}^{=}\psi \mid \psi \,\mathrm{U}^{=}\, \psi \mid \psi \mathrm{S}^{=} \psi \end{aligned}$$

Here, $p \in \mathcal{P}$, $a \in \mathcal{V}$, $\delta \in \mathbb{Z}$. Intuitively, the quantifier $\mathrm{C}_{@a}\psi$ restricts the evaluation of ψ to the class word induced by attribute a at the current position.

Next we define the formal semantics of position formulas. Let w be an attributed word and i a position on w:

- $w, i \models p$ if $p \in w[i].\mathcal{P}$;
- $w, i \models \neg\varphi$ if $w, i \not\models \varphi$;
- $w, i \models \varphi_1 \vee \varphi_2$ if $w, i \models \varphi_1$ or $w, i \models \varphi_2$;
- $w, i \models \mathrm{X}\varphi$ if $i + 1 \leq |w|$ and $w, i + 1 \models \varphi$;
- $w, i \models \varphi_1 \mathrm{U} \varphi_2$ if there exists a $j \geq i$ such that $w, j \models \varphi_2$ and $w, j' \models \varphi_1$ for all $j' \in [i, j)$;
- $w, i \models \mathrm{C}^{\delta}_{@a}\psi$ if $w[i].@a \neq nil$, $i + \delta \in [1, |w|]$, and $w, i + \delta, w[i].@a \models \psi$.

The operators Y and S are the past counterparts of X and U respectively. Their semantics is defined analogously[3].

Next, we define the semantics of class formulas. Let w be an attributed word, i a position on w and d a data value.

- $w, i, d \models \varphi$ if $w, i \models \varphi$;
- $w, i, d \models @a$ if $w[i].@a = d$;
- $w, i, d \models \mathrm{X}^{=}\varphi$ if there exists a $j \in pos_d(w)$ with $j > i$, and for the smallest such j it holds $w, j, d \models \varphi$;

[2] As we will use $\mathcal{A}$ for automata we use $\mathcal{V}$ here: **V**ariables.

[3] To avoid ambiguity: $p\mathrm{S}q$ holds if there is a q-position in the past and at the intermediate positions p holds.

- $w, i, d \models \varphi_1 \,\mathrm{U}^{=}\, \varphi_2$ if there exists a $j \in pos_d(w)$ with $j \geq i$ such that $w, j, d \models \varphi_2$ and $w, k, d \models \varphi_1$ for all $k \in pos_d(w) \cap [i, j)$.

For the past class operators Y and S the semantics is defined analogously and the semantics of the Boolean connectors is as usual. Finally, $w \models \varphi$, if $w, 1 \models \varphi$. We denote the set of positional formulas by BD-LTL.

Besides $\bot$ and $\top$ we use the following usual abbreviations:

$$\mathrm{F}\varphi := \top \mathrm{U} \varphi \quad \mathrm{G}\varphi := \neg \mathrm{F} \neg \varphi \quad \mathrm{P}\varphi := \top \mathrm{S} \varphi \quad \mathrm{H}\varphi := \neg \mathrm{P} \neg \varphi$$

The abbreviations $\mathrm{F}^{=}$ and $\mathrm{G}^{=}$ and their past counterparts are defined analogously. Furthermore, we abbreviate $\mathrm{C}^{\delta}_{@a} @b$ by $@a = \mathrm{X}^{\delta} @b$.

2.3 Example: a simple client/server scenario

The following example illustrates how properties can be expressed in BD-LTL.

Consider an internet platform that uses m servers $S_1, \ldots, S_m$ to process queries from clients. Every client shall have a unique client number. As we do not know beforehand how many clients will use the platform, we model the client numbers by the set $\mathcal{D} = \mathbb{N}$.

Each of the servers can either idle, be queried by a client or serve the answer for a query. For server j, the actions are modeled by the set of propositions $\{q_j, s_j, i_j\}$. Runs of the internet platform can now be represented by an attributed word with attribute set $\mathcal{ATT} = \{S_1, \ldots, S_m\}$ and set of propositions $\bigcup_{1 \leq j \leq m} \{q_j, s_j, i_j\}$. That a server S_j shall perform exactly one action from $\{q_j, s_j, i_j\}$ at any given time, can be easily expressed by a BD-LTL-formula.

Let us look at an example system with three servers A, B and C. An example run represented as an attributed word could look as follows.

Pos	1	2	3	4	5	6
Props	$\{q_A, q_B, i_C\}$	$\{q_A, q_B, q_C\}$	$\{s_A, q_B, s_C\}$	$\{s_A, s_B, i_C\}$	$\{i_A, s_B, q_C\}$	$\{i_A, s_B, s_C\}$
A	1	2	2	1	–	–
B	2	3	4	2	3	4
C	–	1	1	–	2	2

Here, e.g., at position 5 server A is idling, server B is serving client 3 and server C is queried by client 2. Properties of runs can be expressed by BD-LTL formulas:

- Queries are always served and a client can query a second time on a server only after the previous query has been served:

$$\bigwedge_{Z \in \{A,B,C\}} \mathrm{G}(q_Z \to \mathrm{C}_{@Z}(\mathrm{X}^{=}(@Z \to \neg q_Z)\, \mathrm{U}^{=}(@Z \wedge s_Z)))$$

- A server Z can serve a client only if there is an unanswered query by that client (i.e. the last action by that client on Z was a query):

$$\bigwedge_{Z \in \{A,B,C\}} \mathrm{G}(s_Z \to \mathrm{C}_{@Z}(\mathrm{Y}^{=}(\neg @Z)\mathrm{S}^{=}(@Z \wedge q_Z))))$$

- A client with an open query on server A shall only be allowed to query server C until server A answered the query:

$$G(q_A \to \mathrm{C}_{@A}(\neg @B \wedge \mathrm{X}^{=}((\neg(q_A \wedge A) \wedge \neg(q_B \wedge B))\, \mathrm{U}^{=}\, s_A)))$$

3 Expressiveness of BD-LTL

In this section we will give a short overview of established logics on strings with data values and outline how BD-LTL fits in. We give a short introduction to freeze LTL and CLTL$^\diamond$, see [12] and [10] for more details. Afterwards we compare these two logics to BD-LTL.

3.1 BD-LTL versus LTL$_1^\downarrow$

Freeze LTL is an extension of LTL for data words by a freeze quantifier that binds the data value of the current position to a variable (aka register) and allows to compare the value of a position with the value bound to a variable. Satisfiability for freeze LTL is undecidable even for two registers [12], therefore [12] proposed the 1-register fragment LTL$_1^\downarrow$. In the framework of 1-attributed words, formulas of LTL$_1^\downarrow$ are of the form

$$\varphi ::= p \mid \downarrow\varphi \mid\uparrow\mid \neg\varphi \mid \varphi\wedge\varphi \mid \mathrm{X}\varphi \mid \mathrm{Y}\varphi \mid \varphi\mathrm{U}\varphi \mid \varphi\mathrm{S}\varphi.$$

The formal semantics of LTL$_1^\downarrow$ (on data strings) can be found in [12]. We illustrate it by a simple example: the formula $\mathrm{G}(p \to \downarrow\mathrm{F}(q\wedge\uparrow))$ expresses that each p-position has a future q-position with the same data value.

In [12], the fragment simple LTL$_1^\downarrow$ was invented, where at most one temporal operator is allowed between the the freeze quantifier $\downarrow$ and a value test $\uparrow$. Furthermore, only the unary temporal operators $\mathrm{X}^k, \mathrm{Y}^k, \mathrm{X}^k\mathrm{F}, \mathrm{Y}^k\mathrm{P}$, $k\in\mathbb{N}$ are allowed. Here, $\mathrm{X}^k\mathrm{F}$ is considered a single operator, that is $\downarrow\mathrm{X}^k\mathrm{F}\uparrow$ is an allowed formula. The relative expressive power of BD-LTL and LTL$_1^\downarrow$ can be summarized in the following two propositions.

▶ Proposition 1. Every property of 1-attributed words that is expressible in BD-LTL can also be expressed in LTL$_1^\downarrow$.

The statement also holds for all extensions of BD-LTL considered in Section 5. Note however, that LTL$_1^\downarrow$ is undecidable whereas BD-LTL and its main extension in Section 5 are decidable.

▶ Proposition 2. The following logics are equivalent on 1-attributed words

(i) Simple LTL$_1^\downarrow$

(ii) BD-LTL without Until and Since extended by $\mathrm{F}^\delta_{\neq}$ and $\mathrm{P}^\delta_{\neq}$.

Here, $\mathrm{F}^\delta_{\neq}\varphi$ intuitively navigates to a future position of distance $\geq\delta$ with a different data value and evaluates φ there. In the notation of Section 5 it is an abbreviation for $\top\mathrm{U}^\delta_{@a}(\overline{@a}\wedge\varphi)$. Note, that an analogous operator $F^\delta_{=}\varphi$ for equal data values can be simulated by $\mathrm{C}^\delta_{@a}F^{=}\varphi$. The proof of both propositions is straightforward and therefore omitted.

3.2 BD-LTL versus CLTL$^\diamond$

Temporal logic of repeating values (CLTL$^\diamond$) was introduced in [10]. CLTL$^\diamond$-formulas are of the form $\varphi ::= x = X^\delta y \mid x = \diamond y \mid \varphi\wedge\varphi \mid \neg\varphi \mid \mathrm{X}\varphi \mid \varphi\mathrm{U}\varphi \mid \mathrm{Y}\varphi \mid \varphi\mathrm{S}\varphi$, where x, y are from a set of variables. A CLTL$^\diamond$-formula with variables $\{x_1,\dots,x_m\}$ is evaluated on sequences of m-tuples of data values (without labels from a finite set) but the extension to $\{x_1,\dots,x_m\}$-complete attributed strings is straightforward. A formula $x = X^\delta y$ tests whether component x of the current position has the same data value as component y of the δ-next position. A formula $x = \diamond y$ is true if there is a (strict) future position with the same data value on component y as the current position has on component x. The semantics of all other operators is as usual. The following proposition is straightforward, since $x = \diamond y$ and $x = X^\delta y$ can be encoded by $\mathrm{C}^0_{@x}\mathrm{X}^{=}\mathrm{F}^{=}@y$ and $\mathrm{C}^\delta_{@x}@y$, respectively.

▶ Proposition 3. On $\{x_1,\dots,x_m\}$-complete attributed words BD-LTL is strictly more expressive than CLTL$^\diamond$.

4 Decidability of Basic Data LTL

This section states the main decidability result for BD-LTL and undecidability results for some of its extensions.

▶ **Theorem 1.** *Satisfiability for BD-LTL is decidable.*

The proof of this result proceeds in two main steps. First it is shown that the satisfiability problem for arbitrary attributed words can be reduced to the case of 1-attributed words. A similar reduction from the multi-attribute to the 1-attribute case (for a different logic) has been given in [13]. For 1-attributed words, BD-LTL-formulas can be translated into data automata [5] and thus the satisfiability problem for BD-LTL can be reduced to the decidable non-emptiness problem for data automata.

In a nutshell, a *data automaton* $\mathcal{A} = (\mathcal{B}, \mathcal{C})$ consists of a finite state transducer $\mathcal{B}$ (the *base automaton*) and a finite state automaton $\mathcal{C}$ (the *class automaton*). The string projection of a given 1-attributed word w is processed by the base automaton, firstly. Then the output w' of $\mathcal{B}$ is processed class-wise by the class automaton, i.e. $\mathcal{C}$ is run for every data value d on the class word $class_d(w)$. $\mathcal{A}$ accepts w, if $\mathcal{B}$ accepts and $\mathcal{C}$ accepts all class words.

We give only a proof sketch, see the full version of this article for a detailed proof [17].

▶ **Theorem 2.** *Satisfiability for BD-LTL on 1-attributed words is decidable.*

Proof. (Sketch.)

Let φ be a BD-LTL formula over a proposition set $\mathcal{P}$ and the attribute set $\{a\}$.

In the following we often call 1-attributed words simply *words*. Our automata will expect instead of words w over P extended words w' with additional propositions. First, w' allows the subformulas of φ as propositions. The intention is that a position i of w' is marked with ψ if and only if $w, i \models \psi$. Furthermore, we use propositions $=_r$ for every $r \in \{-N, \ldots, -1, 1, \ldots, N\}$, for some N that is at least as large as every δ occurring in φ. Proposition $=_r$ shall hold at position i if and only if $w[i].@a = w[i+r].@a$.

The data automaton $\mathcal{A}$ now checks whether those additional propositions are correct. $\mathcal{A}$ is the intersection of data automata for the following conditions:

i) The propositions $=_r$ are placed correctly.
ii) Subformulas are placed correctly (i.e. position i is labeled with proposition ψ if and only if ψ is fulfilled on position i).
iii) φ is placed on the first position

Condition iii) can be easily checked. Condition i) can be checked by a data automaton [3].

For ii), a data automaton for every subformula ψ is constructed, assuming the correctness of subformulas of ψ. Checking the correctness is straightforward for subformulas ψ of type p, $\neg\chi$, $\chi \vee \chi$, $\mathrm{X}\chi$, $\mathrm{Y}\chi$, $\chi\mathrm{U}\chi$, $\chi\mathrm{S}\chi$. Basically, these formulas can be checked solely by the base automaton. The construction is equally straightforward for all types of class formulas. In these cases, basically only class automata are needed.

To deal with the δ-shift in formulas of the form $\mathrm{C}^{\delta}_{@a}\psi$ we use the propositions $=_r$. E.g., to validate propositions of the form $\psi = \mathrm{C}^{7}_{@a}\mathrm{F}^{=}\chi$ at position i, the class automaton $\mathcal{A}_\psi$ infers from the $=_r$ propositions how many positions the class word has between i and $i+7$, then it skips these positions and starts searching for a χ-position from there.

◀

Theorem 1 can be easily extended to the case of attributed attributed ω-words as in [5].

▶ **Theorem 3.** *Satisfiability for BD-LTL on attributed ω-words is decidable.*

Extensions of BD-LTL quickly yield undecidability. We consider two such extensions here.

BD-LTL with Navigation along Tuples. We extend $\mathrm{C}_{@a}$ to a quantifier $\mathrm{C}_{@a,@b}$ that 'freezes' the values d_a and d_b of the attributes a and b, respectively. Operators $\mathrm{X}^{=}, \mathrm{Y}^{=}, \mathrm{U}^{=}$ and $\mathrm{S}^{=}$ in the scope of $\mathrm{C}_{@a,@b}$ then move along positions that have attributes with data values d_a and d_b. At such positions the values of tuples of attributes can be tested for equality with (d_a, d_b). For example the property 'there is a future position with proposition p where attribute c carries the same data value as attribute a at the current position, likewise for d and b' can be expressed by $\mathrm{C}_{@a,@b} F^{=}((@c, @d) \wedge p)$.

However, already a restricted version of this extension is undecidable. We consider the operators $\mathrm{X}_{@\mathrm{a},@\mathrm{b}}$ and $\mathrm{Y}_{@\mathrm{a},@\mathrm{b}}$. Let the semantics of $\mathrm{X}_{@\mathrm{a},@\mathrm{b}}$ be defined by $w, i \models \mathrm{X}_{@\mathrm{a},@\mathrm{b}}\varphi$ if there is a $j > i$ with $w[i].@a = w[j].@a$ and $w[i].@b = w[j].@b$ and for the smallest such j it holds $w, j \models \varphi$. The operator $\mathrm{Y}_{@a,@b}$ is defined analogously.

▶ **Theorem 4.** *BD-LTL extended by the operators $\mathrm{X}_{@a,@b}$ and $\mathrm{Y}_{@a,@b}$ is undecidable on finite (or infinite) attributed words.*

The proof is along the lines of Proposition 27 in [5] by a reduction from the Post Correspondence Problem (PCP).

BD-LTL with From-Now-On Operator. The *from-now-on*-operator N introduced in [18] restricts the range of past operators. For an attributed word $w = w_1 \dots w_n$ and a position i of w let $suf_i(w) := w_i \dots w_n$ be the suffix of w starting at position i. The semantics of N is then defined by

- $w, i \models \mathrm{N}\varphi$ if $suf_i(w), 1 \models \varphi$

▶ **Theorem 5.** *BD-LTL extended by the operator* N *is undecidable on finite (or infinite) attributed words.*

The proof is by a reduction from the non-emptiness problem for Minsky two counter automata [21].

5 Extended Navigation

As already discussed before, the navigational abilities of BD-LTL are limited. It seemingly cannot[4] even express the simple property that for every p-position i there is a q-position $j > i$ such that $w[j].@b \neq w[i].@a$. Furthermore, in class formulas $\rho \mathrm{U}^{=} \tau$, the formula ρ can only refer to positions of the current class. Of course, it would be desirable to allow more general forms of "Until navigation".

In this section we discuss different possibilities to extend the navigational abilities of BD-LTL in an "Until fashion", some of which are decidable and some undecidable. In particular, we exhibit an U-operator with the ability to navigate to a position with a different attribute value and to state some properties on (all) intermediate positions and show that BD-LTL remains decidable with this extension. The property stated in the previous paragraph can be expressed using this operator.

The extensions we study allow formulas of the type $\rho \mathrm{U}^{\delta}_{@a} \tau$, where $\delta \geq 0$. Intuitively, this operator "freezes" the current value of attribute a and searches for a position j such that τ

[4] We did not attempt to find a proof for this statement as we were aiming for an extended logic, anyway. However, we did not find a simple way to express the property.

holds at j and ρ hold everywhere in $[i+\delta, j)$. In formulas as above, we will refer to ρ as the *intermediate formula* and τ as the *target formula*. The "shift" parameter δ is needed as we aim to design a semantic extension of simple $\mathrm{LTL}_1^{\downarrow}$.

Syntactically, the formulas ρ and τ are positive Boolean combinations of position formulas and positive and negative attribute tests. More formally, we define the syntax of *U-subformulas* χ by $\chi ::= \varphi \mid @b \mid \overline{@b} \mid \chi \vee \chi \mid \chi \wedge \chi$. Intuitively, negative attribute tests $\overline{@b}$ check that attribute b has a value (!) that is different from the current frozen value.

Thus, the semantics of formulas $\rho \mathrm{U}^{\delta}_{@a} \tau$, where ρ and τ are U-subformulas, is defined by the following additional rules.

- $w, i \models \rho \mathrm{U}^{\delta}_{@a} \tau$ if there exists a $j \geq i + \delta$ such that $w, j, w[i].@a \models \tau$ and $w, k, w[i].@a \models \rho$ for all $k \in [i+\delta, j)$
- $w, i, d \models \overline{@b}$ if $w[i].@b \notin \{nil, d\}$.

We simply use $\mathrm{U}_{@a}$ instead of $\mathrm{U}^{0}_{@a}$. We remark that $\rho \mathrm{U}^{-\delta}_{@a} \tau$, for $\delta \geq 0$ can be expressed by $(\rho \,\mathrm{U}_{@a}\, \tau \wedge \bigwedge_{i=1}^{\delta} \rho_i) \vee (\bigvee_{j=1}^{\delta} (\tau_j \wedge \bigwedge_{i=j+1}^{\delta} \rho_i))$, where, for $k \in [1, \delta]$, ρ_k and τ_k are obtained from ρ and τ, respectively, by replacing every position formula φ by $\mathrm{Y}^k \varphi$, every $@b$ by $@a = Y^k @b$ and every $\overline{@b}$ by $\neg @a = Y^k @b$. It can be observed that this formula has the intended meaning (that is, the semantics obtained by using $-\delta$ in the above semantics definition). $\rho \mathrm{S}_{@a} \tau$ is defined analogously.

First of all, we will see that the above mentioned restriction for class formulas $\rho \mathrm{U}^{=} \tau$ is indeed crucial. More precisely, if we allow positive attribute tests in the target formula of a formula $\rho \,\mathrm{U}_{@a}\, \tau$ then the logic becomes undecidable even if the intermediate formulas are restricted to position formulas.

▶ **Theorem 6.** *Let $\mathcal{L}$ denote the extension of BD-LTL by the formation rule $\varphi ::= \chi \,\mathrm{U}_{@a}\, \chi$, where χ denotes* U*-subformulas such that*

- *all intermediate formulas are position formulas and*
- *all target formulas are of the form $@a \wedge \varphi$ with a position formula φ.*

Then, satisfiability of $\mathcal{L}$ on finite (or infinite) attributed words is undecidable. This holds even for 1-attributed words.

The proof is again by a reduction from the non-emptiness problem for Minsky two counter automata [21]. As Theorem 6 does not leave much room for extensions of $\mathrm{U}_{@a}$ operators with positive attribute tests in the target formula we focus on negative attribute tests in target formulas. However, as $\rho \mathrm{U}^{\delta}_{@a}(\tau_1 \vee \tau_2) \equiv (\rho \mathrm{U}^{\delta}_{@a} \tau_1) \vee (\rho \,\mathrm{U}_{@a}\, \tau_2)$ and position formulas are closed under conjunctions it is clearly sufficient to consider target formulas of the form $\varphi \wedge \overline{@b_1} \wedge \cdots \wedge \overline{@b_k}$. Unfortunately, at this point our techniques can only deal with the case $k = 1$.

We turn our attention now to the intermediate formulas ρ. We recall that in the case of positive attribute tests in target formulas even position formulas as intermediate formulas yield undecidability. In the case of (single) negative attribute tests in target formulas we can allow arbitrary intermediate position formulas.

Furthermore, we can add positive and negative attribute tests, but only in a limited way. More precisely, we define the logic XD-LTL by adding $\varphi ::= \chi \mathrm{U}^{\delta}_{@a} \chi' \mid \chi \mathrm{S}^{\delta}_{@a} \chi'$, to the formation rules of BD-LTL and requiring that

1. χ is restricted to formulas of the form $\rho \vee (@b \wedge \rho^{=}) \vee (\overline{@b} \wedge \rho^{\neq})$ where $\rho^{=}, \rho^{\neq}$ are position formulas and $\rho^{\neq}$ logically implies[5] $\rho^{=}$, and

[5] Readers who prefer a syntactical criterion might think of a formula $\rho^{=}$ of the form $\varphi \vee \rho^{\neq}$.

2. χ' is restricted to formulas of the form $\overline{@b} \wedge \tau$, where τ is a position formula.

Intuitively, $\rho^{=}$ constrains positions where $@b$ equals the current value of $@a$ whereas $\rho^{\neq}$ constrains those where it does not. The requirement that $\rho^{\neq}$ implies $\rho^{=}$ is needed for the proof of Theorem 8.

Clearly XD-LTL strictly extends BD-LTL and is contained in $\mathrm{LTL}_1^{\downarrow}$. Further it strictly extends two-variable logic on 1-attributed words.

Following the general idea of the decidability proof for BD-LTL we first show decidability of satisfiability for 1-attributed words and reduce the general case to this one.

▶ **Theorem 7.** *Satisfiability for XD-LTL on finite 1-attributed words is decidable.*

Proof. (Sketch.) The proof basically extends the proof of Theorem 2 for formulas of type $\psi = (@a \wedge \rho^{=}) \vee (\overline{@a} \wedge \rho^{\neq}) \mathrm{U}^{\delta}_{@a} (\overline{@a} \wedge \tau)$. Note that in the case of 1-attributed words, any additional disjunct ρ in the intermediate formula can be pushed into the disjunction by or-ing it with both $\rho^{=}$ and $\rho^{\neq}$.

For a given position i with data value d fulfilling $w, i \models \psi$ we call the minimal position j that fulfills $\rho^{\neq}$ and has a data value different from d, the *ψ-shepherd for i*. We write $H(j)$ for the herd of j, that is the set of positions for which j is a ψ-shepherd. With each τ-position j we associate a set $S(j)$ of *special positions*. Roughly speaking, if i is in the herd of j, then positions in $[i, j)$ with the same data value as i are special. The special interval $I(j)$ for a shepherd j is the minimal interval containing $S(j)$. Two crucial observations are that (1) all positions in $S(j)$ have the same data value and (2) $|I(j) \cap I(j')| \leq \delta$ for $j \neq j'$.

In a nutshell, the idea for the construction of the data automaton for ψ is as follows. Besides the propositions for the subformulas, we use further propositions of the form H, e^{+} and e^{-} with the intention that for each shepherd marked by τ, the end points of the special interval are marked by e^{+} and e^{-}, respectively, and all positions in $H(j)$ are marked by H.

As we are testing satisfiability, we can safely assume that all those propositions are already present in the input word, but their consistency has to be verified by the automaton. The automaton then checks that for each τ-position j the corresponding e^{+}- and e^{-}-positions are as intended. Further it guesses and checks all other positions in $S(j)$. Finally consistency of H- and τ-positions is verified.

As for BD-LTL, special attention is needed for $\delta \neq 0$. For the detailed proof, we refer the reader to the full version of the paper [17].

◀

By a straightforward extension of the proof of Theorem 1 we get the following.

▶ **Theorem 8.** *Satisfiability for XD-LTL on finite attributed words is decidable.*

6 Conclusion

We conclude by stating some questions that should be investigated further. We would be interested to understand the exact border of undecidability. At this point, it is not exactly clear which kinds of intermediate and target formulas can be allowed for $\mathrm{U}^{\delta}_{@a}$. It would also be interesting to compare our logics with other logics that can deal with values, particularly with guarded LTL-FO of [14]. Further investigations could try to identify fragments with more reasonable complexity and try to add more arithmetics to the data domain.

References

1 R. Alur and T. A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

2 T. Arons, A. Pnueli, S. Ruah, J. Xu, and L. D. Zuck. Parameterized verification with automatically computed inductive assertions. In *CAV*, volume 2620 of *Lecture Notes in Computer Science*, pages 221–234, 2001.

3 H. Björklund and T. Schwentick. On notions of regularity for data languages. *Theor. Comput. Sci.*, 411(4-5):702–715, 2010.

4 M. Bojanczyk. Personal communication, 2006.

5 M. Bojanczyk, A. Muscholl, T. Schwentick, L. Segoufin, and C. David. Two-variable logic on words with data. In *LICS*, pages 7–16. IEEE Computer Society, 2006.

6 A. Bouajjani, P. Habermehl, Y. Jurski, and M. Sighireanu. Rewriting systems with data. In *Fundamentals of Computation Theory*, volume 4639 of *Lecture Notes in Computer Science*, pages 1–22. Springer Berlin / Heidelberg, 2007.

7 P. Bouyer. A logical characterization of data languages. *Inf. Process. Lett.*, 84(2):75–85, 2002.

8 P. Bouyer, A. Petit, and D. Therien. An algebraic approach to data languages and timed languages. *Inf. Comput.*, 182(2):137–162, 2003.

9 S. Demri. LTL over integer periodicity constraints. In *FoSSaCS*, pages 121–135, 2004.

10 S. Demri, D. D'Souza, and R. Gascon. A decidable temporal logic of repeating values. In S. N. Artëmov and A. Nerode, editors, *LFCS*, volume 4514 of *Lecture Notes in Computer Science*, pages 180–194. Springer, 2007.

11 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. In *LICS '06: Proceedings of the 21st Annual IEEE Symposium on Logic in Computer Science*, pages 17–26, Washington, DC, USA, 2006. IEEE Computer Society.

12 S. Demri and R. Lazic. LTL with the freeze quantifier and register automata. *ACM Trans. Comput. Log.*, 10(3), 2009.

13 S. Demri, R. Lazić, and D. Nowak. On the freeze quantifier in constraint LTL: Decidability and complexity. *Inf. Comput.*, 205(1):2–24, 2007.

14 A. Deutsch, R. Hull, F. Patrizi, and V. Vianu. Automatic verification of data-centric business processes. In *ICDT*, pages 252–267, 2009.

15 D. Figueira and L. Segoufin. Future-looking logics on data words and trees. In R. Královic and D. Niwinski, editors, *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 331–343. Springer, 2009.

16 M. Kaminski and N. Francez. Finite-memory automata. *Theor. Comput. Sci.*, 134(2):329–363, 1994.

17 A. Kara, T. Schwentick, and T. Zeume. Temporal logics on words with multiple data values. Available from arXiv:1010.1139, 2010.

18 F. Laroussinie and P. Schnoebelen. A hierarchy of temporal logics with past. *Theor. Comput. Sci.*, 148(2):303–324, 1995.

19 A. Lisitsa and I. Potapov. Temporal logic with predicate lambda-abstraction. In *TIME 2005*, pages 147–155, 2005.

20 A. Manuel. Two orders and two variables. In *MFCS*, volume 6281 of *Lecture Notes in Computer Science*, pages 513–524, 2010.

21 M. L. Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1967.

22 F. Neven, T. Schwentick, and V. Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. Comput. Log.*, 5(3):403–435, 2004.

23 T. Schwentick and T. Zeume. Two-variable logic with two order relations. In *CSL*, volume 6247 of *Lecture Notes in Computer Science*, pages 499–513, 2010.

24 T. Tan. On pebble automata for data languages with decidable emptiness problem. In *MFCS*, volume 5734 of *Lecture Notes in Computer Science*, pages 712–723, 2009.

First-Order Logic with Reachability Predicates on Infinite Systems

Stefan Schulz

Lehrstuhl für Informatik 7, RWTH Aachen, 52056 Aachen, Germany

Abstract

This paper focuses on first-order logic (FO) extended by reachability predicates such that the expressiveness and hence decidability properties lie between FO and monadic second-order logic (MSO): in FO(R) one can demand that a node is reachably from another by some sequence of edges, whereas in FO(Reg) a regular set of allowed edge sequences can be given additionally. We study FO(Reg) logic in infinite grid-like structures which are important in verification. The decidability of logics between FO and MSO on those simple structures turns out to be sensitive to various parameters. Furthermore we introduce a transformation for infinite graphs called set-based unfolding which is based on an idea of Lohrey and Ondrusch. It allows to transfer the decidability of MSO to FO(Reg) onto the class of transformed structures. Finally we extend regular ground tree rewriting with a skeleton tree. We show that graphs specified in this way coincide with those expressible by vertex replacement and product operators. This allows to extend decidability results of Colcombet for FO(R) to those graphs.

Keywords and phrases First-Order Logic, Reachability, Infinite Grid, Structure Transformation, Unfolding, Ground Tree Rewriting, Vertex Replacement with Product

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.493

1 Introduction

The general task in verification is to check whether an infinite graph structure satisfies a given specification, which is usually expressed by a logical formula in the fundamental first-order (FO) or monadic second-order (MSO) logic. These logics are well-studied and over the years many classes of infinite structures have been identified where the theory of one of these logics is decidable. The most prominent examples are automatic structures [13, 15, 2] like Presburger arithmetic $(\mathbb{N}, 0, 1, +)$ for FO, and natural numbers with successor $(\mathbb{N}, \mathrm{S})$ by Büchi [3] and the binary tree $(\{0,1\}^*, \mathrm{S}_1, \mathrm{S}_2)$ by Rabin [20] for MSO. In verification one often specifies properties dealing with reachability in graph structures. These cannot be expressed in FO logic. One could switch to MSO logic which comes at the expense of worse decidability properties. To overcome this problem we consider FO logic extended by reachability predicates. In FO(R) logic these predicates express that some element is reachable from another by using a subset of the available edge relations. In FO(Reg) logic one can express reachability by sequences of edge relations which form a regular language. Both logics lie strictly between FO and MSO according to their expressiveness and decidability.

In Section 3 we mainly study the decidability of FO(Reg) logic on infinite grids. Although they look simple one can express strong properties by formulas which makes them interesting for verification. Furthermore FO logic is known to be decidable whereas MSO logic is undecidable. We consider an n-dimensional grid to be a structure having $\mathbb{N}^n$ as domain and a successor and predecessor relation for each dimension. The decidability of FO(Reg) logic turns out to be sensitive to the various parameters, which is mostly inherited from important, closely related formalisms like Petri nets, vector addition systems, pushdown automata, register machines, and logic over arithmetic. Furthermore we extend studies of

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 493–504

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Wöhrle and Thomas [21] about FO logic extended by an operator for transitive closure, which reveals interesting parallels to the situation of FO(Reg) logic.

A generic way of generating structures having some particular decidable logic is by structure transformation. A standard transformation is interpretation where a new structure is defined by specifying its domain and relations by formulas. It preserves the decidability of FO or MSO logic, respectively, when the formulas come from this logic. Another approach for graph structures is unfolding where the new structure is the tree of all finite paths starting in a given initial vertex in the original structure. This transformation preserves the decidability of MSO logic [10]. In Section 4 we define set-based unfolding which is an abstraction of unfolding. It does not preserve the decidability of full MSO logic but FO(Reg) logic is decidable anyhow. The idea for this transformation goes back to a construction of Lohrey and Ondrusch [18]. Thus set-based unfolding is such a type of transformation, which maps the decidability of one particular logic to the decidability of a weaker logic. Another example of such a transformation is finite set interpretation by Colcombet and Löding [9] which maps decidability of weak MSO logic to FO logic in the resulting structure.

Lastly in Section 5 we extending an equivalence result for a class of graphs where FO(R) logic is decidable. The first way to describe this class is by regular ground tree rewriting (RGTR) systems, where the domain is given as a set of finite trees and edge relations are induced by rewriting rules which replace a subtree by another. The decidability of FO(R) logic on RGTR graphs was shown by a transformation to tree automata and tree transducers [11]. The second formalism is completely differently motivated and describes graphs by operations on colored graphs: the vertex replacement and product (VRP) operators. Usually those operators are aligned in a (possibly infinite) tree, called the VRP tree, and specify the graph which is its least fixed point. Colcombet [7, 8] showed RGTR graphs to be equivalent to graphs represented by regular VRP trees. He furthermore showed the FO(R) theory to be decidable for graphs of VRP trees with decidable MSO theory. With this motivation we extend RGTR to regular skeleton ground tree rewriting (RSGTR) by adding a usually infinite skeleton tree and obtain the equivalence to graphs of arbitrary VRP trees. The transformation furthermore preserves decidability of MSO logic of the VRP tree and skeleton tree, which makes the FO(R) theory of RSGTR graphs decidable if the skeleton has a decidable MSO theory.

2 Preliminaries

We use the following notations for intervals of integers: $\mathbb{Z} := (-\infty, \infty)$ and $\mathbb{N} := [0, \infty)$. By $\wp(S)$ we denote the powerset of a set S. Let Σ be an alphabet, i.e., a finite set of symbols, then Σ^* is the set of words over Σ, i.e., finite sequences of its symbols, and a language is a set of words. The number of occurrences of a symbol $\sigma \in \Sigma$ in a word w is $|w|_\sigma \in \mathbb{N}$, the length of w is $|w| \in \mathbb{N}$ and the empty word ϵ is the word of length $|\epsilon| = 0$. We assume the reader to be familiar with regular languages, i.e., the languages specified by regular expressions or equivalently by finite automata.

A structure $\mathcal{A} = \big(A, (f_i)_{i\in\mathfrak{f}}, (R_i)_{i\in\mathfrak{R}}\big)$ consists of a (possibly infinite) domain A, functions $f_i : A^{n_i} \to A$ and relations $R_i \subseteq A^{m_i}$ each of arity n_i and m_i, respectively. A *relational* structure has only relations, and it is a *graph structure* if all relations are of arity 1 or 2. We consider only structures with finitely many functions and relations. An equivalence relation $\sim \subseteq A \times A$ is a congruence on a relational structure $\mathcal{A} = \big(A, (R_i)_{i\in\mathfrak{R}}\big)$ if $R_i(x_1, \ldots, x_{m_i}) \Leftrightarrow R_i(y_1, \ldots, y_{m_i})$ for all R_i and $x_1 \sim y_1, \ldots, x_{m_i} \sim y_{m_i}$. Its quotient $\mathcal{A}/\!\sim = \big(A', (R'_i)_{i\in\mathfrak{R}}\big)$ is defined as $A' := \big\{\, [x] \;\big|\; x \in A \,\big\}$ and $\big([x_1], \ldots, [x_{m_i}]\big) \in R'_i :\Leftrightarrow (x_1, \ldots, x_{m_i}) \in R_i$ with

$[x] := \{ y \mid y \sim x \}$.

With first-order (FO) logic one can specify properties of a structure by using terms of variables $(x, y, z, \ldots)$ and functions (f_i), comparing terms $(=, R_i)$, quantifying elements $(\exists, \forall)$ and its boolean combinations $(\neg, \wedge, \vee, \rightarrow)$. Monadic second-order (MSO) logic additionally allows quantification over element sets $(X, Y, Z, \ldots)$ and using them as unary relations. Weak MSO (WMSO) logic is a variant which only quantifies over finite sets. The theory of a logic and a structure is the set of formulas of that logic which have no free variables and hold in the structure. The property that an element can be reached from another in some graph structure can be expressed in MSO and WMSO logic but not in FO logic. For a graph structure $\mathcal{G} = \big(V, (P_\gamma)_{\gamma\in\Gamma}, (E_\sigma)_{\sigma\in\Sigma}\big)$ we define FO(Reg) logic to be the extension of FO logic by regular reachability predicates $\mathsf{reach}_L(x, y)$, for variables x, y and a regular language $L \subseteq \Sigma^*$ (finitely represented by a regular expression or finite automaton), meaning that position y can be reached from position x by some sequence of edges $E_{\sigma_1}, \ldots, E_{\sigma_n}$ such that $\sigma_1 \cdots \sigma_n \in L$. Let FO(R) logic be its restriction to simple reachability predicates $\mathsf{reach}_{\Sigma_0^*}(x, y)$ with $\Sigma_0 \subseteq \Sigma$. Reachability predicates can be expressed in MSO and WMSO logic by induction on the operators of a regular expression for the language of the predicate. It uses the fact that the transitive closure is expressible in MSO and WMSO. The expressiveness of the above logics increases as follows:

$$\text{FO} \quad \leq \quad \text{FO(R)} \quad \leq \quad \text{FO(Reg)} \quad \leq \quad \begin{cases} \text{MSO} \\ \text{WMSO} \end{cases}$$

Note that WMSO is usually a sublogic of MSO since finiteness is MSO-definable in most standard structures. FO(Reg) logic over $\mathcal{G}$ can furthermore be identified with FO logic over $\big(V, (P_\gamma)_{\gamma\in\Gamma}, (E_L)_{L\subseteq\Sigma^*, L \text{ regular}}\big)$, i.e., reachability is considered only with respect to the edge relations instead of arbitrary FO-definable relations (analogous for FO(R) logic).

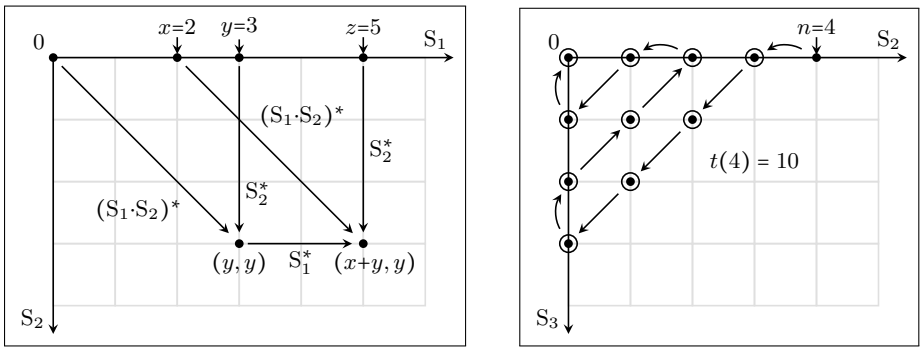
3 Reachability in Infinite Grids

We consider the n-dimensional infinite grid $\mathcal{N}^n := \big(\mathbb{N}^n, (\mathrm{S}_i, \bar{\mathrm{S}}_i)_{1\leq i\leq n}\big)$ to be the n-th product of the natural numbers $\mathcal{N} = (\mathbb{N}, \mathrm{S}, \bar{\mathrm{S}})$ with successor and predecessor, i.e., S_i and $\bar{\mathrm{S}}_i$ are the successor and predecessor relation of dimension i. The MSO theory is known to be decidable for $\mathcal{N}$ [3] but undecidable for its products. By using Büchi's result about the decidability of Presburger arithmetic, i.e., the FO theory of $(\mathbb{N}, 0, 1, +)$, one can easily see that the FO(R) theory is decidable for grids $\mathcal{N}^n$ of any dimension n. For this reason we study the situation for FO(Reg) logic, which reveals an interesting phenomenon.

▸ **Theorem 1.** *The FO(Reg) theory of $\mathcal{N}^2$ is decidable.*

Proof. We reduce this theory to Presburger arithmetic. To this end we transform a given FO(Reg) formula over $\mathcal{N}^2$ into an equivalent Presburger formula by interpreting each grid position $x = (x_1, x_2) \in \mathbb{N}^2$ as numbers $x_1, x_2 \in \mathbb{N}$. Then the standard FO operators can be transformed straightforward. It remains to express a reachability predicate $\mathsf{reach}_L(x, y)$ as Presburger formula, which we will do by use of vector addition systems with states (VASS).

A 2-dimensional VASS $\mathcal{V} = (Q, \Delta)$ consists of a finite state set Q and a finite transition relation $\Delta \subseteq Q \times \mathbb{Z}^2 \times Q$. For states $p, q \in Q$ we define the reachability relation $R_{p,q} \subseteq (\mathbb{N}^2)^2$ consisting of all $(x, y) \in (\mathbb{N}^2)^2$ such that there exist sequences $z_0, \ldots, z_k \in \mathbb{N}^2$ and $(q_0, d_0, q_1), \ldots, (q_{k-1}, d_k, q_k) \in \Delta$ with $x = z_0$, $z_k = y$, $p = q_0$, $q_k = q$ and $z_i = z_{i-1} + d_i$ for all $i \in \{1, \ldots, k\}$. For a reachability predicate $\mathsf{reach}_L(x, y)$ with language $L = L(\mathcal{A})$ of some automaton $\mathcal{A} = \big(Q, \{\mathrm{S}_1, \bar{\mathrm{S}}_1, \mathrm{S}_2, \bar{\mathrm{S}}_2\}, \delta, q_0, F\big)$ we define the VASS (Q, Δ) with $(p, e_i, q) \in \Delta$ iff



(a) Geometric addition $x + y = z$

(b) Geometric multiplication simulated by the function $t(n) := n \cdot (n+1) \div 2$

Figure 1 Arithmetic in the grid by: simulating addition and multiplication

$\delta(p, \mathrm{S}_i) = q$, and $(p, -e_i, q) \in \Delta$ iff $\delta(p, \bar{\mathrm{S}}_i) = q$ where $e_1 := (1,0)$ and $e_2 := (0,1)$. It is obvious from the construction, that for $x, y \in \mathbb{N}^2$ in the grid $\mathsf{reach}_L(x,y)$ holds iff $(x,y) \in R_{q_0,q_f}$ for some $q_f \in F$. Now we can make use of a result from Leroux and Sutre [16], stating that for any two states p, q of a 2-dimensional VASS the reachability relation $R_{p,q} \subseteq (\mathbb{N}^2)^2$ is semi-linear (when identifying $(\mathbb{N}^2)^2$ with $\mathbb{N}^4$), i.e., a finite union of linear sets. Hence, $\bigcup_{q_f \in F} R_{q_0,q_f}$ is the relation defined by $\mathsf{reach}_L(x,y)$, which is still semi-linear (over $\mathbb{N}^4$). This finishes the proof since semi-linear sets are effectively equivalent to the sets definable in Presburger arithmetic [12]. ◂

If we consider the simpler model of the grid $\mathcal{N}_+^n = \left(\mathbb{N}^n, (\mathrm{S}_i)_{1 \le i \le n}\right)$ with only successor relations S_i, then the FO(Reg) theory is decidable for any dimension. This can be proven similarly to the above theorem by reducing a reachability predicate $\mathsf{reach}_L(x,y)$ to its Parikh image $\left\{ (|w|_{\mathrm{S}_1}, \ldots, |w|_{\mathrm{S}_n}) \mid w \in L \right\} \subseteq \mathbb{N}^n$, i.e., the tuples of occurrences of each symbol in words of L, which is effectively semi-linear for any n [19]. On the other hand the proof of Theorem 1 cannot be extended to dimension 3 as in this case the semi-linearity is not present any longer [14]. The next result shows the sharpness of Theorem 1.

▸ **Theorem 2.** *The FO(Reg) theory of $\mathcal{N}^3$ is undecidable.*

Proof. We reduce the FO arithmetic, i.e., the FO theory of $(\mathbb{N}, 0, 1, +, \cdot)$, which is known to be undecidable, to the considered theory. Thus we transform a given FO formula over the arithmetic into an equivalent FO(Reg) formula over $\mathcal{N}^3$ by encoding a number $n \in \mathbb{N}$ as grid position $(n,0,0) \in \mathbb{N}^3$. W.l.o.g. we treat the arithmetic as relational, i.e., with relations $0, 1 \subseteq \mathbb{N}^1$ and $+, \cdot \subseteq \mathbb{N}^3$. It is easy to transform the standard FO operators as well as the relations 0 and 1. Thus only the relations + and $\cdot$ are remaining.

The addition $x + y = z$ can be defined geometrically in the grid as motivated in Fig. 1a by finding intersection points along horizontal, vertical and diagonal lines:

$$\psi_+(x,y,z) := \exists y', z' \Big(\mathsf{reach}_{(\mathrm{S}_1 \cdot \mathrm{S}_2)^*}(0,y') \wedge \mathsf{reach}_{\mathrm{S}_2^*}(y,y') \wedge \mathsf{reach}_{(\mathrm{S}_1 \cdot \mathrm{S}_2)^*}(x,z') \wedge \mathsf{reach}_{\mathrm{S}_2^*}(z,z') \wedge \mathsf{reach}_{\mathrm{S}_1^*}(y',z') \Big).$$

We reduce the multiplication to addition and the triangle function $t(n) := n \cdot (n+1) \div 2$ since $x \cdot y = z$ iff $t(x+y) = t(x) + t(y) + z$. Figure 1b geometrically motivates $t(n) = \left| \{ (i,j) \in \mathbb{N}^2 \mid i + j < n \} \right|$ as the number of positions in the triangle below $(n,0)$ in the plane. It further shows a path along the counterdiagonals with length exactly $t(n)$. To lift the input

from $(n,0,0)$ to the starting point $(0,n,0)$ in the plane of the second and third dimension, we have to swap the position of the input $n \in \mathbb{N}$, which is done by a path of the language $L' := (\bar{S}_1 \cdot S_2)^*$. Then the counterdiagonal path can be described by the language

$$L'' := \left((S_1 \cdot \bar{S}_2) \cdot (S_1 \cdot \bar{S}_2 \cdot S_3)^* + (S_1 \cdot \bar{S}_3) \cdot (S_1 \cdot S_2 \cdot \bar{S}_3)^*\right)^*$$

where the first dimension is used to count the length of the path. Now we can define $t(n)$ to be the maximal first component that is reachable by $L := L' \cdot L''$ from $(n,0,0)$:

$$\psi_t(x,y) \;:=\; \mathsf{reach}_L(x,y) \wedge \forall y' \left(\mathsf{reach}_L(x,y') \to \mathsf{reach}_{(S_1+\bar{S}_2+\bar{S}_3)^*}(y',y)\right).$$

This guarantees that the path turns only at border positions, i.e., from $x = (n,0,0)$ one reaches $(0,n,0)$ by L' and then $y = \big(t(n),0,0\big)$ by L''. For the correctness it is obvious that $n \le t(n)$ for all $n \in \mathbb{N}$, and that taking a shortcut in L' or L'' yields a smaller result. ◂

By a reduction to the FO arithmetic, we showed the FO(Reg) theory of $\mathcal{N}^3$ to be *highly undecidable* as well: each set of the arithmetical hierarchy (which consists of the sets definable in FO arithmetic) can be reduced to it. This fact makes it surprising that the same logic is decidable in the 2-dimensional case (cf. Theorem 1) anyhow. The hardness is introduced by the limitation of natural numbers, i.e., the boundedness in one direction. One can easily show the FO(Reg) theory of the grid $\mathcal{Z}^n := \big(\mathbb{Z}^n, (S_i, \bar{S}_i)_{1 \le i \le n}\big)$ to be decidable by using Parikh images again.

We end this section with a digression on $\mathrm{FO}(\mathrm{TC})_{(1)}$, i.e., FO logic extended by an operator for the transitive closure (TC) of $\mathrm{FO}(\mathrm{TC})_{(1)}$-definable binary relations. Here we consider two variants which are powerful enough: $\mathrm{FO}(\mathrm{TC})^1_{(1)}$ is FO logic with TC only for FO-definable relations, and $\mathrm{FO}(\mathrm{TC})^2_{(1)}$ is FO logic with TC only for $\mathrm{FO}(\mathrm{TC})^1_{(1)}$-definable relations, i.e., the TC operator can not be nested, or at most once, respectively. Their expressiveness stays below MSO and WMSO on grid structures (without proof):

$$\mathrm{FO} \;\le\; \mathrm{FO(R)} \;\le\; \left\{\begin{array}{c}\mathrm{FO(Reg)}\\ \mathrm{FO}(\mathrm{TC})^1_{(1)}\end{array}\right\} \;\le\; \mathrm{FO}(\mathrm{TC})^2_{(1)} \;\le\; \mathrm{FO}(\mathrm{TC})_{(1)} \;\le\; \left\{\begin{array}{c}\mathrm{MSO}\\ \mathrm{WMSO}\end{array}\right.$$

Wöhrle and Thomas [21] studied the decidability of these logics on the 2-dimensional grid. They showed the $\mathrm{FO}(\mathrm{TC})^1_{(1)}$ theory of $\mathcal{N}^2$ to be decidable by a reduction to Presburger arithmetic, and the $\mathrm{FO}(\mathrm{TC})^2_{(1)}$ theory of $\mathcal{N}^2$ to be highly undecidable by reducing FO arithmetic. By copying the proof of Theorem 2 one can furthermore show the $\mathrm{FO}(\mathrm{TC})^1_{(1)}$ theory of $\mathcal{N}^3$ to be highly undecidable as well. It is interesting to observe the same phenomenon that the logic changes from being decidable to highly undecidable between dimension 2 and 3.

4 Set-Based Unfolding

The unfolding (or unraveling) $\mathsf{Unf}_{v_0}(\mathcal{G})$ of a graph structure $\mathcal{G}$ is a tree, the vertices of which are finite paths of $\mathcal{G}$ starting at the initial vertex v_0. The relations are inherited from $\mathcal{G}$: unary relations are set according to the last vertex of a path, and edge relations are used to extend paths, i.e., following an edge of the last vertex. Courcelle and Walukiewicz [10] have shown that $\mathsf{Unf}_{v_0}(\mathcal{G})$ preserves the decidability of MSO logic of $\mathcal{G}$ for any initial vertex v_0 that is MSO-definable in $\mathcal{G}$.

We present a model-theoretic structure transformation which is similar to the unfolding and preserves some logic decidability too. Instead of having finite paths as domain, we abstract each such path $v_0 E_{\sigma_1} v_1 \dots E_{\sigma_n} v_n$ to the *trace* $\big(v_n, \{v_0, v_1, \dots, v_n\}\big)$, in which v_n is the last vertex and $\{v_0, v_1, \dots, v_n\}$ is the (finite) set of visited vertices of the path:

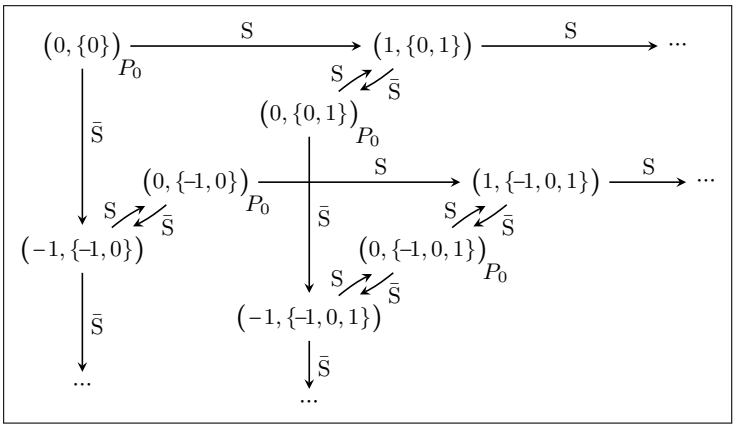


Figure 2 Set-based unfolding $\mathsf{SetUnf}_0(\mathcal{Z})$ of the structure $\mathcal{Z} = (\mathbb{Z}, P_0, \mathrm{S}, \bar{\mathrm{S}})$

▸ **Definition 3.** Let $\mathcal{G} = \big(V, (P_\gamma)_{\gamma\in\Gamma}, (E_\sigma)_{\sigma\in\Sigma}\big)$ be a graph structure with unary and binary relations P_γ and E_σ, respectively. For some set of initial traces $I \subseteq V \times \wp(V)$, i.e., for each $(v, V_0) \in I$: V_0 is finite and $v \in V_0$, the *set-based unfolding* $\mathsf{SetUnf}_I(\mathcal{G})$ *of* $\mathcal{G}$ *from* I is the graph structure $\big(V', (P'_\gamma)_{\gamma\in\Gamma}, (E'_\sigma)_{\sigma\in\Sigma}\big)$ with

1. domain $V' \subseteq V \times \wp(V)$ being the smallest set of traces which contains I, and for all σ, v, v', V_0 if $(v, V_0) \in V'$ and $(v, v') \in E_\sigma$ then $\big(v, V_0 \cup \{v'\}\big) \in V'$,
2. predicates $(v, V_0) \in P'_\gamma$ iff $v \in P_\gamma$, and
3. edges $\big((v, V_0), (v', V'_0)\big) \in E'_\sigma$ iff $(v, v') \in E_\sigma$ and $V'_0 = V_0 \cup \{v'\}$.

We abbreviate $\mathsf{SetUnf}_{v_0}(\mathcal{G})$ for $\mathsf{SetUnf}_{\{(v_0,\{v_0\})\}}(\mathcal{G})$ with the initial trace $(v_0, \{v_0\})$ representing some v_0 in $\mathcal{G}$. Note that $\mathsf{SetUnf}_{v_0}(\mathcal{G})$ may not be a tree in contrast to $\mathsf{Unf}_{v_0}(\mathcal{G})$:

▸ **Example 4** (Free group and free inverse monoid)**.** Consider the graph structure $\mathcal{Z} = (\mathbb{Z}, P_0, \mathrm{S}, \bar{\mathrm{S}})$ with unary relation for 0, successor and predecessor relation, which is isomorphic to the free group $\mathrm{FG}(\{\mathrm{S}\})$ of the singleton alphabet $\{\mathrm{S}\}$. Its set-based unfolding $\mathsf{SetUnf}_0(\mathcal{Z})$ from vertex 0 yields the structure depicted in Fig. 2, which is isomorphic to the free inverse monoid $\mathrm{FIM}(\{\mathrm{S}\})$ of the same alphabet.

▸ **Theorem 5.** *The FO(Reg) theory of a quotient* $\mathsf{SetUnf}_I(\mathcal{G})/{\sim}$ *is decidable, if the MSO theory of the graph* $\mathcal{G}$ *is decidable and the set* I *of initial traces, the congruence* $\sim$*, as well as finiteness are MSO-definable in* $\mathcal{G}$*.*

Proof. This proof is based on one from Lohrey and Ondrusch [18]. It goes by interpretation, i.e., each FO(Reg) formula φ over $\mathsf{SetUnf}_I(\mathcal{G})/{\sim}$ can be transformed effectively into an equivalent MSO formula $\hat{\varphi}$ over $\mathcal{G}$. Thereby each trace of $\mathsf{SetUnf}_I(\mathcal{G})$ is encoded in $\mathcal{G}$ by a tuple (x, X) of a position and a set variable. Given formulas $\psi_I(x, X)$ for the initial traces, and $\psi_\sim(x, X, y, Y)$ for the congruence, we define the transformation $\hat{\varphi}$ of φ inductively:

$$\widehat{x = y} := \psi_\sim(x, X, y, Y) \qquad \widehat{\neg\varphi} := \neg\hat{\varphi}$$
$$\widehat{P_\gamma(x)} := P_\gamma(x) \qquad \widehat{\varphi \vee \psi} := \hat{\varphi} \vee \hat{\psi}$$
$$\widehat{E_\sigma(x, y)} := \widehat{\mathsf{reach}_\sigma(x, y)} \qquad \widehat{\exists x\, \varphi} := \exists x\, \exists X \left(\hat{\varphi} \wedge \psi_{\mathrm{dom}}(x, X)\right)$$
$$\widehat{\mathsf{reach}_L(x, y)} := \exists y'\, \exists Y', Z \left(\mathsf{Reach}_L(x, y', Z) \wedge (Y' = X \cup Z) \wedge \psi_\sim(y, Y, y', Y') \wedge \psi_{\mathrm{dom}}(y', Y')\right)$$

where $\psi_{\mathrm{dom}}(x,X) := \exists y\, \exists Y,Z \left(\psi_I(y,Y) \wedge \mathsf{Reach}_{\Sigma^*}(y,x,Z) \wedge (X = Y \cup Z)\right)$ is the domain formula, and $\mathsf{Reach}_L(x,y,Z)$ is an extended reachability predicate stating that y is reachable from x by a path labeled by some word in L and exactly visiting the vertices Z, which was shown to be MSO-definable [18] since finiteness is MSO-definable in $\mathcal{G}$. The transformation of the reachability predicate is correct since the congruence $\sim$ also applies to finite paths, and hence reachability, i.e., (x,X) reaches (y,Y) by some edge sequence iff (x',X') reaches (y',Y') by the very same sequence for all traces $(x,X) \sim (x',X'), (y,Y) \sim (y',Y')$. ◂

The main result is a simpler version of this theorem with equality as trivial congruence:

▸ **Corollary 6.** *The FO(Reg) theory of* $\mathsf{SetUnf}_{v_0}(\mathcal{G})$ *is decidable if the MSO theory of the graph* $\mathcal{G}$ *is decidable and the vertex* v_0*, as well as finiteness are MSO-definable in* $\mathcal{G}$*.*

This reads similar to the preservation of MSO-decidability for unfolding [10] although it is a weaker result. On the other hand Example 4 demonstrates the sharpness of Corollary 6. The structure $\mathcal{Z} = (\mathbb{Z}, P_0, \mathrm{S}, \bar{\mathrm{S}})$ has a decidable MSO theory [3] and finiteness is MSO-definable in $\mathcal{Z}$. Thus $\mathsf{SetUnf}_0(\mathcal{Z})$ has a decidable FO(Reg) theory whereas the MSO theory is undecidable (by interpretation of the infinite grid [4]). Note that the results of this section also apply to WMSO, respectively. A good usage of set-based unfolding would be the Caucal hierarchy [6], which is a strict hierarchy of graph structures obtained by alternately unfolding and MSO-interpreting the class of finite graphs[1], since all such graphs have decidable MSO and WMSO theories [6, 17].

5 Regular Ground Tree Rewriting with Skeleton

We are dealing with trees over a *ranked alphabet* Σ, i.e., each symbol $f \in \Sigma$ has a certain arity or rank $|f| \in \mathbb{N}$. A tree $t = f(t_1, \ldots, t_{|f|})$ has a symbol $f \in \Sigma$ at its root and each t_i is a tree again. We can view t as a (partial) function, which maps its domain $\mathrm{dom}(t) := \{\epsilon\} \cup \bigcup_{1 \le i \le |f|} \left(i \cdot \mathrm{dom}(t_i)\right) \subseteq \mathbb{N}^*$ to Σ with $t(\epsilon) := f$ and $t(i \cdot w) := t_i(w)$ for $1 \le i \le |f|$. The subtree $t|_v$ of t at a position $v \in \mathrm{dom}(t)$ is defined as $t|_v(w) = t(v \cdot w)$. A tree is infinite if its domain is infinite, and it is furthermore *regular* if it has only finitely many different subtrees. We can view t also as a graph structure $\left(\mathrm{dom}(t), (E_i)_{1 \le i \le |f|, f \in \Sigma}, (P_f)_{f \in \Sigma}\right)$ with $E_i := \left\{ (w, w \cdot i) \mid w \cdot i \in \mathrm{dom}(t) \right\}$ and $P_f := \left\{ w \mid t(w) = f \right\}$. T_Σ (T_Σ^{fin}) denotes the set of (finite) trees over Σ. Subsets of T_Σ are called tree languages. Analogous to words we use automata to specify languages of finite trees. A *tree automaton* $\mathcal{A} = \left(Q, \Sigma, (\Delta_f)_{f \in \Sigma}, F\right)$ consists of a finite state set Q with some accepting states $F \subseteq Q$, a finite ranked alphabet Σ and transition relations $\Delta_f \subseteq Q \times Q^{|f|}$ for each $f \in \Sigma$. A finite tree $t \in T_\Sigma^{\mathsf{fin}}$ is in the tree language $T(\mathcal{A})$ recognized by $\mathcal{A}$ if there exists a run $\rho : \mathrm{dom}(t) \to Q$, which is accepting, i.e., $\rho(\epsilon) \in F$, and which respects Δ, i.e., $\left(\rho(r), \rho(r \cdot 1), \ldots, \rho(r \cdot |f|)\right) \in \Delta_f$ for each position $r \in \mathrm{dom}(t)$ with $t(r) = f$. We call $\mathcal{A}$ *bottom-up deterministic* if for all $f \in \Sigma$, $(q_1, \ldots, q_{|f|}) \in Q^{|f|}$ there is at most one $q \in Q$ with $(q, q_1, \ldots, q_{|f|}) \in \Delta_f$. $\mathcal{A}$ is *top-down deterministic* if $|F| = 1$ and for all $f \in \Sigma$, $q \in Q$ there is at most one $(q_1, \ldots, q_{|f|}) \in Q^{|f|}$ with $(q, q_1, \ldots, q_{|f|}) \in \Delta_f$. Tree languages recognizable by a bottom-up deterministic (top-down deterministic) tree automata are called *regular* (*top-down deterministic recognizable*).

A regular ground tree rewriting (RGTR) system $(T, \Sigma, A, \to)$ consists of a top-down deterministic recognizable domain $T \subseteq T_\Sigma^{\mathsf{fin}}$ and finitely many rewriting rules $L \xrightarrow{a} R$ with

[1] In [6] Caucal actually defined the hierarchy with inverse rational mappings instead of MSO interpretation, which is shown to be equivalent [5].

regular tree languages $L, R \subseteq T_\Sigma^{\mathsf{fin}}$ and label $a \in A$. It defines a graph structure with T as vertices, and a-labeled edges between trees $t, t' \in T$ if there is some rule $L \xrightarrow{a} R$ such that t' is obtained by replacing one subtree $l \in L$ in t by $r \in R$. Furthermore there is a constant for each tree. Dauchet and Tison [11] have shown that the FO(R) theory is decidable for RGTR graphs $(T, \Sigma, A, \to)$ with complete domain $T = T_\Sigma^{\mathsf{fin}}$ and rules of the form $\{l\} \xrightarrow{a} \{r\}$ with only singletons. The proof uses a translation to tree transducers and tree-automatic relations, such that one can actually extend the proof to general RGTR graphs. This result cannot be extended to higher logics like FO(Reg) or $\mathrm{FO(TC)}^1_{(1)}$ since we showed these theories to be undecidable on $\mathcal{N}^3$ and grids are some of the simplest structures representable by ground tree rewriting:

▸ **Example 7** (Infinite grid)**.** Let $(T, \Sigma, A, \to)$ be an RGTR system over the ranked alphabet $\Sigma = \{2, 1, 0_L, 0_R\}$, each symbol having the arity according to its number, with $T = \left\{ 2(1^x(0_L), 1^y(0_R)) \;\middle|\; x, y \in \mathbb{N} \right\}$, labels $A = \{\mathrm{S}_1, \mathrm{S}_2\}$, and rewriting rules $0_L \xrightarrow{\mathrm{S}_1} 1(0_L)$, $0_R \xrightarrow{\mathrm{S}_2} 1(0_R)$. Its graph is isomorphic to the infinite grid $\mathcal{N}^2_+ = (\mathbb{N}^2, \mathrm{S}_1, \mathrm{S}_2)$.

An algebraically motivated way of specifying infinite graphs is *vertex replacement with product* (VRP) [7, 8] where graphs are the least fixed point of equations of operations on colored graphs. These are structures $G = \big(V, (P_c)_{c\in C}, (E_a)_{a\in A}\big)$ with (possibly infinite) domain of colored vertices $V = \biguplus_{c\in C} P_c$ and edge relations E_a for a finite set C of colors and A of actions. The family of those graphs is called $\mathcal{G}_{A,C}$. For the following let us fix some sets actions A and colors C. The five VRP operators on colored graphs are

1. *Constant singleton graph*: $\dot{c} : {\mathcal{G}_{A,C}}^0 \to \mathcal{G}_{A,C}$, just one vertex having color $c \in C$,
2. *Recoloring*: $[\phi] : {\mathcal{G}_{A,C}}^1 \to \mathcal{G}_{A,C}$, for some color mapping $\phi : C \to C$,
3. *Adding edges*: $[c \overset{a}{\bowtie} d] : {\mathcal{G}_{A,C}}^1 \to \mathcal{G}_{A,C}$, labeled by $a \in A$ between colors $c, d \in C$,
4. *Disjoint union*: $\oplus : {\mathcal{G}_{A,C}}^2 \to \mathcal{G}_{A,C}$, and
5. *Asynchronous product*: $\otimes_\eta : {\mathcal{G}_{A,C}}^2 \to \mathcal{G}_{A,C}$, for a function $\eta : C^2 \to C$ merging the colors[2].

To specify an infinite graph we use a (possibly infinite) term of VRP operators called *VRP tree*, i.e., a tree over the ranked alphabet $\Omega_{A,C}$ consisting of the VRP operators $\dot{c}$, $[\phi]$, $[c \overset{a}{\bowtie} d]$, $\oplus$, $\otimes_\eta$ with arities 0, 1, 1, 2, 2, respectively. The *interpretation* $[\![t]\!]$ of a VRP tree $t \in T_{\Omega_{A,C}}$ is defined as its least fixed point according to the subgraph relation[3]. This is a complete partial order with the empty graph as least element, which guarantees the existence of a unique least fixed point $[\![t]\!]$ that furthermore is equal to the supremum of the chain $[\![t]\!]_0 \subseteq [\![t]\!]_1 \subseteq \cdots$ where $[\![t]\!]_d$ is the partial interpretation up to depth $d \in \mathbb{N}$, i.e., $[\![t]\!]_0 = \bot$ (the empty graph) and $[\![f(t_1, \ldots, t_n)]\!]_{d+1} = f([\![t_1]\!]_d, \ldots, [\![t_n]\!]_d)$ with VRP operator $f \in \Omega_{A,C}$.

▸ **Example 8** (Infinite grid)**.** Let $A = \{\mathrm{S}_1, \mathrm{S}_2\}$, $C = \{0, 1, 2\}$, and the VRP tree t as follows (depicted in Fig. 3):

$$t \;:=\; t_1 \otimes_{[(c,d)\mapsto 0]} t_2, \qquad t_i \;:=\; [0 \overset{\mathrm{S}_i}{\bowtie} 1]\Big(\dot{0} \oplus \begin{bmatrix} 0\mapsto 1 \\ 1\mapsto 2 \\ 2\mapsto 2 \end{bmatrix} t_i\Big) \quad \text{for } i \in \{1, 2\}.$$

The interpretation $[\![t]\!]$ of t is isomorphic to $\mathcal{N}^2_+ = (\mathbb{N}^2, \mathrm{S}_1, \mathrm{S}_2)$ when ignoring colors.

Colcombets main results are, that interpretations of regular VRP trees are effectively equivalent to RGTR graphs (up to isomorphism and color removal), and that the FO(R) theory of an interpretation is decidable if the VRP tree has a decidable MSO theory [7]. We

[2] The asynchronous product $\otimes_\eta$ has a fixed function η in [7]; and is called $\square_\eta$ in [8].
[3] $\big(V, (P_c)_{c\in C}, (E_a)_{a\in A}\big) \subseteq \big(V', (P'_c)_{c\in C}, (E'_a)_{a\in A}\big)$ if $V \subseteq V'$, $E_a \subseteq E'_a$, and $P_c \subseteq P'_c$ for each $a \in A, c \in C$.

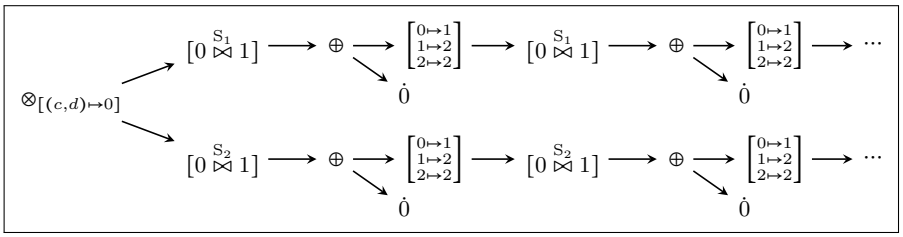


Figure 3 VRP tree defining the infinite grid $\mathcal{N}_+^2 = (\mathbb{N}^2, S_1, S_2)$

are going to remove the regularity demand from the equivalence result by making RGTR as powerful as VRP trees in general. To this end we equip RGTR with a (usually infinite) tree, which functions as a skeleton for the specification of the domain and the rewriting rules. This concept is based on the *overlay* $t\|_s$ of trees $t : \mathrm{dom}(t) \to \Sigma$ and $s : \mathrm{dom}(s) \to \Gamma$ with

$$t\|_s : \ \mathrm{dom}(t) \to \Sigma\|_\Gamma, \qquad t\|_s(w) \ := \ \begin{cases} \big(t(w), s(w)\big) & \text{if } w \in \mathrm{dom}(s), \\ \big(t(w), \bot\big) & \text{otherwise} \end{cases}$$

with *overlay alphabet* $\Sigma\|_\Gamma := \Sigma \times \big(\Gamma \uplus \{\bot\}\big)$ of rank $|(f,g)| := |f|$. We write $T\|_s := \big\{ t\|_s \ \big|\ t \in T \big\}$.

▸ **Definition 9.** A *regular skeleton ground tree rewriting* (RSGTR) system $(s, \Gamma, T, \Sigma, A, \to)$ consists of a *skeleton* $s \in T_\Gamma$, i.e., a tree over the ranked alphabet Γ, such that $(T, \Sigma\|_\Gamma, A, \to)$ forms an RGTR system. The graph structure it defines is the graph of the RGTR system $(T, \Sigma\|_\Gamma, A, \to)$ restricted to $T_\Sigma^{\mathsf{fin}}\|_s$, i.e., trees where the overlaid component corresponds to the skeleton tree $s \in T_\Gamma$.

▸ **Theorem 10.** *VRP interpretations are effectively equivalent to RSGTR graphs (up to isomorphism and color removal). Furthermore the conversion between the VRP tree and the skeleton tree preserves the decidability of MSO logic and regularity.*

Proof. The first part is the direction from VRP trees to RSGTR systems. Consider a given VRP tree $t \in T_{\Omega_{A,C}}$. We can simulate it by an RSGTR system (depending only on A and C) with t as skeleton. From the definition of the interpretation via chains, it follows that each node of the interpretation is represented by exactly one *finite prefix* of t. This is a part of t starting from the root, such that for each vertex with label $f \in \Omega_{A,C}$ of the prefix:

1. if $f \in \big\{[\phi], [c \overset{a}{\bowtie} d]\big\}$ then the (unique) child has to belong to the prefix as well,
2. if $f = \oplus$ then either the left or the right child belongs to the prefix, and
3. if $f = \otimes_\eta$ then both children belong to the prefix.

Figure 4 depicts (when ignoring the numbers below the vertices) such a finite prefix of the infinite VRP tree of Fig. 3. Those prefixes form a deterministic top-down recognizable set when using the overlay alphabet $\Sigma\|_{\Omega_{A,C}}$ with $\Sigma := \{0, 1, 2, 2_L, 2_R\}$, each symbol having its number as rank. Then use 0 for constants, 1 for unary operators, 2 for products, and $2_L, 2_R$ for the left and right branches of disjoint unions, respectively.

To simulate the edges as introduced by the edge adding operators of t we have to look at the coloring of vertices. The colors of a prefix can be computed easily in a bottom-up manner for each subtree such that the color of each subtree corresponds to the color of the vertex which is VRP-represented by that very subtree. Starting by copying the constant colors at the leaves, the computation simply merges at each subtree the colors of its children according to the semantic of the considered operator. In Fig. 4 the colors are written next to each vertex. If a subtree is labeled by an operator $[c \overset{a}{\bowtie} d]$ and has color c assigned to it then

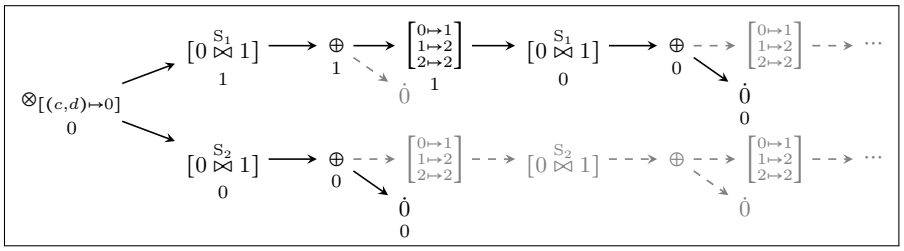


Figure 4 A finite prefix of the VRP tree of Fig. 3 representing grid position $(1,0)$

it can be rewritten to another one of color d. In Fig. 4 this means that there would be an S_2-labeled transition from the subtree with operator $[0 \overset{S_2}{\bowtie} 1]$ and color 0 to another subtree of color 1, which can only lead to the prefix where the paths at both children are of the same length, i.e., grid position $(1,1)$. This can be implemented by regular rewriting rules over the overlay alphabet $\Sigma\|_{\Omega_{A,C}}$. We skip the exact definition, since it is easy but purely technical and does not bring any deeper insight than the explanation above. The MSO-decidability and regularity are preserved trivially since we chose t itself to be the skeleton.

Showing the converse is a bit more challenging. Consider a given RSGTR system $(s, \Gamma, T, \Sigma, A, \Delta)$. To finish the proof we define a transformation of s into a VRP tree t whose structure mimics s and T in a top-down manner, whereas the definition of Δ is simulated bottom-up in its colors. Let T be specified by a deterministic top-down tree automaton $\mathcal{A} = \big(Q, \Sigma\|_\Gamma, (\delta_f)_{f\in\Sigma\|_\Gamma}, q_0\big)$, and let Δ be represented by both the deterministic bottom-up tree automaton $\mathcal{A}'_q = \big(Q', \Sigma\|_\Gamma, (\delta'_f)_{f\in\Sigma\|_\Gamma}, \{q\}\big)$ and the relation $\Delta' \subseteq Q' \times A \times Q'$, such that $l \in L, r \in R$ for some rule $L \xrightarrow{a} R$ of Δ iff $l \in T(\mathcal{A}_p), r \in T(\mathcal{A}_q)$ for some $(p,a,q) \in \Delta'$. This alternative representation of the transitions can be obtained by a product construction of the automata in the rules of Δ. The transformation mentioned above is the composition of the following tree transformations, each of which has the desired preservation properties:

1. For simplicity we first extend $s \in T_\Gamma$ to an infinite m-ary tree $s' \in T_{\Gamma'}$ where $m := \max\big\{\, |f| \;\big|\; f \in \Sigma \,\big\}$ is the maximal rank of Σ, each symbol of $\Gamma' := \Gamma \uplus \{\bot\}$ has rank m, and $s'(w) := s(w)$ if $w \in \mathrm{dom}(s)$, or $s'(w) := \bot$ otherwise. Then $t\|_s = t\|_{s'}$ for each $t \in T_\Sigma$.
2. The actual work is done by transforming $s' \in T_{\Gamma'}$ into a *relaxed* VRP tree $t' \in T_{\Omega'_{A,C}}$ where $\Omega'_{A,C}$ is like $\Omega_{A,C}$ but the operators $\oplus$ and $\otimes_\eta$ are lifted to their n-ary correspondents $\bigoplus_{i\in\{1,\dots,n\}}$ and $\bigotimes_\eta$ with $\eta : C^n \to C$ for bounded $n \in \mathbb{N}$. We set actions A as in $\mathcal{A}'$ and the colors $C := Q'$. We transform s' into $t' := \lfloor s' \rfloor_{q_0}$ with $q_0 \in Q$ from $\mathcal{A}$ where $\lfloor s' \rfloor_q$ for each $q \in Q$ and tree $s' = g'(s'_1, \dots, s'_m) \in T_{\Omega'_{A,C}}$ is defined as the (unique) tree
$$\lfloor s' \rfloor_q \;:=\; \underbrace{\cdots[p' \overset{a}{\bowtie} q']\cdots}_{\substack{\text{for each}\\(p',a,q')\in\Delta'}} \;\; \bigoplus_{\substack{\delta_{(f,g')}(q)=\\(p_1,\dots,p_{|f|})}} \left(\otimes_{\delta'_{(f,g')}} \Big(\lfloor s'_1 \rfloor_{p_1}, \dots, \lfloor s'_{|f|} \rfloor_{p_{|f|}} \Big) \right).$$
When ignoring everything that deals with colors in this construction, one can verify that this defines just the domain T by top-down simulating $\mathcal{A}$ with respect to the skeleton s. When just looking at the colors, then they exactly simulate The bottom-up behavior of $\mathcal{A}'$ is exactly simulated by the colors, where we use the relation Δ' to specified by transitions of $\mathcal{A}'$. The way the transformation is defined allows the preservance of regularity and MSO-decidability.
3. Finally we transform $t' \in T_{\Omega'_{A,C}}$ into $t \in T_{\Omega_{A,C'}}$ by simply reducing the n-ary operators $\bigoplus, \bigotimes_\eta$ to their binary versions $\oplus, \otimes_\eta$ or constants (for empty products). In general large products require the introduction of new colors C' for n-tuples of old colors C. ◂

The restriction of Theorem 10 to regular trees, this yields exactly Colcombet's equivalence result [7], since a regular skeleton tree can already be simulated by the domain of an RGTR system. By combining Theorem 10 with the other main result of Colcombet we can lift the decidability of the FO(R) theory from RGTR to RSGTR:

▸ **Corollary 11.** *The FO(R) theory of an RSGTR graph is decidable if the skeleton tree has a decidable MSO theory.*

6 Conclusion

Let us summarize the main results of this work about FO logic extended by reachability. We have classified the decidability of FO(Reg) logic on infinite grids where the boundary of decidability turned out to be between dimension 2 and 3 (Theorems 1 and 2). By set-based unfolding we have introduced a new graph transformation which does not preserve the decidability of MSO logic but still transfers it to a decidable FO(Reg) logic on the unfolded graph (Corollary 6). By extending RGTR systems with a skeleton tree we have given an automaton-based formalism with the same expressive power as VRP trees (Theorem 10). One can furthermore reduce FO(Reg) logic on the graphs of those systems to MSO logic on its skeleton tree (Corollary 11).

Besides these results there still remain open questions. From graphs with decidable MSO theory we can generate members of the family of graphs having decidable FO(Reg) theories by set-based unfolding. Although not proven, we suppose that this family contains more graphs than obtainable in this way. And if so, how can this family be characterized? Furthermore it is known that interpretations of regular VRP trees are equivalent to RGTR graphs [7] whereas interpretations of regular VR trees (without the product operation) are equivalent to prefix recognizable graphs [1]. Which subclass of RSGTR graphs is described by interpretations of (possibly irregular) VR trees with decidable MSO theory?

Acknowledgements With thanks to Arnaud Carayol and Christof Löding for introducing me to these topics and constantly sharing their valuable ideas.

References

1 Klaus Barthelmann. When can an equational simple graph be generated by hyperedge replacement? In *MFCS*, volume 1450, pages 543–552, 1998.

2 Achim Blumensath and Erich Grädel. Automatic structures. In *LICS*, pages 51–62. IEEE Computer Society, 2000.

3 J. R. Büchi. On a decision method in restricted second-order arithmetic. In E. Nagel, P. Suppes, and A. Tarski, editors, *Int. Congr. for Logic, Methodology and Philosophy of Science*, pages 1–11. Stanford University Press, 1962.

4 Hugues Calbrix. La théorie monadique du second ordre du monoïde inversif libre est indécidable. *Bulletin of the Belg. Math. Soc.*, 4(1):53–65, 1997.

5 Arnaud Carayol and Stefan Wöhrle. The caucal hierarchy of infinite graphs in terms of logic and higher-order pushdown automata. In *FSTTCS*, volume 2914 of *LNCS*, pages 112–123, 2003.

6 Didier Caucal. On infinite terms having a decidable monadic theory. In *MFCS*, volume 2420 of *LNCS*, pages 165–176. Springer, 2002.

7 Thomas Colcombet. On families of graphs having a decidable first order theory with reachability. In *ICALP*, volume 2380 of *LNCS*, pages 98–109. Springer, 2002.

8 Thomas Colcombet. *Propriétés et représentation de structures infinies*. PhD thesis, Université Rennes I, March 2004.

9 Thomas Colcombet and Christof Löding. Transforming structures by set interpretations. *LMCS*, 3(2):1–36, 2007.

10 Bruno Courcelle and Igor Walukiewicz. Monadic second-order logic, graph converings and unfoldings of transition systems. *Ann. Pure Appl. Logic*, 92(1):35–62, 1998.

11 M. Dauchet and S. Tison. The theory of ground rewrite systems is decidable. In *LICS*, pages 242–248. IEEE Computer Society, 1990.

12 Seymour Ginsburg and Edwin H. Spanier. Semigroups, presburger formulas, and languages. *Pacific Journal of Mathematics*, 16(2):285–296, 1966.

13 Bernard R. Hodgson. Décidabilité par automate fini. *Ann. Sci. Math. Québec*, 7(3):39–57, 1983.

14 John E. Hopcroft and Jean-Jacques Pansiot. On the reachability problem for 5-dimensional vector addition systems. *Theoretical Computer Science*, 8:135–159, 1979.

15 Bakhadyr Khoussainov and Anil Nerode. Automatic presentations of structures. In *Logical and Computational Complexity*, volume 960 of *LNCS*, pages 367–392. Springer, 1995.

16 Jérôme Leroux and Grégoire Sutre. On flatness for 2-dimensional vector addition systems with states. In *CONCUR*, volume 3170 of *LNCS*, pages 402–416. Springer, 2004.

17 Christof Löding. Logic and automata over infinite trees, 2009. Habilitation thesis, RWTH Aachen.

18 Markus Lohrey and Nicole Ondrusch. Inverse monoids: Decidability and complexity of algebraic questions. *Inf. Comput.*, 205(8):1212–1234, 2007.

19 Rohit J. Parikh. On context-free languages. *JACM*, 13(4):570–581, 1966.

20 Michael O. Rabin. Decidability of second-order theories and automata on infinite trees. *Transactions of the American Mathematical Society*, 141:1–35, 1969.

21 Stefan Wöhrle and Wolfgang Thomas. Model checking synchronized products of infinite transition systems. In *LICS*, pages 2–11. IEEE Computer Society, 2004.

Generalized Mean-payoff and Energy Games*

Krishnendu Chatterjee[1], Laurent Doyen[2], Thomas A. Henzinger[1], and Jean-François Raskin[3]

1 **IST Austria (Institute of Science and Technology Austria)**
2 **LSV, ENS Cachan & CNRS, France**
3 **Département d'Informatique, Université Libre de Bruxelles (U.L.B.)**

Abstract

In mean-payoff games, the objective of the protagonist is to ensure that the limit average of an infinite sequence of numeric weights is nonnegative. In energy games, the objective is to ensure that the running sum of weights is always nonnegative. Generalized mean-payoff and energy games replace individual weights by tuples, and the limit average (resp. running sum) of each coordinate must be (resp. remain) nonnegative. These games have applications in the synthesis of resource-bounded processes with multiple resources.

We prove the finite-memory determinacy of generalized energy games and show the inter-reducibility of generalized mean-payoff and energy games for finite-memory strategies. We also improve the computational complexity for solving both classes of games with finite-memory strategies: while the previously best known upper bound was EXPSPACE, and no lower bound was known, we give an optimal coNP-complete bound. For memoryless strategies, we show that the problem of deciding the existence of a winning strategy for the protagonist is NP-complete.

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2010.505

1 Introduction

Graph games and multi-objectives. Two-player games on graphs are central in many applications of computer science. For example, in the synthesis problem, implementations are obtained from winning strategies in games with a qualitative objective such as ω-regular specifications [18, 17, 1]. In these applications, the games have a qualitative (boolean) objective that determines which player wins. On the other hand, games with quantitative objective which are natural models in economics (where players have to optimize a real-valued payoff) have also been studied in the context of automated design [19, 10, 20]. In the recent past, there has been considerable interest in the design of reactive systems that work in resource-constrained environments (such as embedded systems). The specifications for such reactive systems are quantitative, and these give rise to quantitative games. In most system design problems, there is no unique objective to be optimized, but multiple, potentially conflicting objectives. For example, in designing a computer system, one is interested not only in minimizing the average response time but also the average power consumption. In this work we study such multi-objective generalizations of the two most widely used quantitative objectives in games, namely, *mean-payoff* and *energy* objectives [11, 20, 6, 3].

Generalized mean-payoff games. A *generalized mean-payoff game* is played on a finite weighted game graph by two players. The vertices of the game graph are partitioned into positions that belong to Player 1 and positions that belong to Player 2. Edges of the graphs are labeled with k-dimensional vectors w of integer values, i.e., $w \in \mathbb{Z}^k$. The game is played as follows. A pebble is placed on a designated initial vertex of the game graph. The game is

* Full proofs are available in [9].

IARCS Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2010).
Editors: Kamal Lodaya, Meena Mahajan; pp. 505–516

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

played in rounds in which the player owning the position where the pebble lies moves the pebble to an adjacent position of the graph using an outgoing edge. The game is played for an infinite number of rounds, resulting in an infinite path through the graph, called a play. The value associated to a play is the mean value in each dimension of the vectors of weights labeling the edges of the play. Accordingly, the winning condition for Player 1 is defined by a vector of integer values $v \in \mathbb{Z}^k$ that specifies a threshold for each dimension. A play is winning for Player 1 if its vector of mean values is at least v. All other plays are winning for Player 2, thus the game is zero-sum. We are interested in the problem of deciding the existence of a finite-memory winning strategy for Player 1 in generalized mean-payoff games. Note that in general infinite memory may be required to win generalized mean-payoff games, but for practical applications such as the synthesis of reactive systems with multiple resource constraints, the generalized mean-payoff games with finite memory is the relevant model. Moreover, they provide the framework for the synthesis of specifications defined by [2, 8], and the synthesis question for such specifications under *regular (ultimately periodic)* words correspond to generalized mean-payoff games with finite-memory strategies.

Generalized energy games. In generalized energy games, the winning condition for Player 1 requires that, given an initial credit $v_0 \in \mathbb{N}^k$, the sum of v_0 and all the vectors labeling edges up to position i in the play is nonnegative, for all $i \in \mathbb{N}$. The decision problem for generalized energy games asks whether there exists an initial credit v_0 and a strategy for Player 1 to maintain the energy nonnegative in all dimensions against all strategies of Player 2.

Contributions. In this paper, we study the strategy complexity and computational complexity of solving generalized mean-payoff and energy games. Our contributions are as follows.
First, we show that generalized energy and mean-payoff games are determined when played with finite-memory strategies, however, they are not determined for memoryless strategies. For generalized energy games determinacy under finite-memory coincides with determinacy under arbitrary strategies (each player has a winning strategy iff he has a finite-memory winning strategy). In contrast, we show for generalized mean-payoff games that determinacy under finite-memory and determinacy under arbitrary strategies do not coincide. Thus with finite-memory strategies these games are determined, they correspond to the synthesis question with ultimately periodic words, and enjoy pleasant mathematical properties like existence of the limit of the mean value of the weights, and hence we focus on the study of generalized mean-payoff and energy games with finite-memory strategies.
Second, we show that under the hypothesis that both players play either finite-memory or memoryless strategies, the generalized mean-payoff game and the generalized energy game problems are equivalent.
Third, our main contribution is the study of the computational complexity of the decision problems for generalized mean-payoff games and generalized energy games, both for finite-memory strategies and the special case of memoryless strategies. Our complexity results can be summarized as follows: (A) For finite-memory strategies, we provide a nondeterministic polynomial time algorithm for deciding negative instances of the problems[1]. Thus we show that the decision problems are in coNP. This significantly improves the complexity as compared to the EXPSPACE algorithm that can be obtained by reduction to Vass (vector addition systems with states) [4]. Furthermore, we establish a coNP lower bound for these problems by reduction from the complement of the 3SAT problem, hence showing that the problem is coNP-complete. (B) For the case of memoryless strategies, as the games are not

[1] Negative instances are those where Player 1 is losing, and by determinacy under finite-memory where Player 2 is winning.

determined, we consider the problem of determining if Player 1 has a memoryless winning strategy. First, we show that the problem of determining if Player 1 has a memoryless winning strategy is in NP, and then show that the problem is NP-hard (i) even when the weights are restricted to $\{-1, 0, 1\}$; or (ii) when the weights are arbitrary and the dimension is 2.

Related works. Mean-payoff games, which are the one-dimension version of our generalized mean-payoff games, have been extensively studied starting with the works of Ehrenfeucht and Mycielski in [11] where they prove memoryless determinacy for these games. Because of memoryless determinacy, it is easy to show that the decision problem for mean-payoff games lies in NP ∩ coNP, but despite large research efforts, no polynomial time algorithm is known for that problem. A pseudo-polynomial time algorithm has been proposed by Zwick and Paterson in [20], and improved in [5]. The one-dimension special case of generalized energy games have been introduced in [6] and further studied in [3] where log-space equivalence with classical mean-payoff games is established.

Generalized energy games can be viewed as games played on Vass (vector addition systems with states) where the objective is to avoid unbounded decreasing of the counters. A solution to such games on Vass is provided in [4] (see in particular Lemma 3.4 in [4]) with a PSPACE algorithm when the weights are $\{-1, 0, 1\}$, leading to an EXPSPACE algorithm when the weights are arbitrary integers. We drastically improve the EXPSPACE upper-bound by providing a coNP algorithm for the problem, and we also provide a coNP lower bound even when the weights are restricted to $\{-1, 0, 1\}$.

2 Generalized Mean-payoff and Energy Games

Well quasi-orders. Let D be a set. A relation $\preceq$ over D is a *well quasi-order*, wqo for short, if the following holds: (a) $\preceq$ is transitive and reflexive; and (b) for all $f : \mathbb{N} \to D$, there exists $i_1, i_2 \in \mathbb{N}$ such that $i_1 < i_2$ and $f(i_1) \preceq f(i_2)$.

▶ **Lemma 1.** *$(\mathbb{N}^k, \leq)$ is well quasi-ordered.*

Multi-weigthed two-player game structures. A *multi-weigthed two-player game structure* is a tuple $G = (S_1, S_2, s_{\mathsf{init}}, E, k, w)$ where $S_1 \cap S_2 = \emptyset$, and S_i $(i = 1, 2)$ is the finite set of *Player i positions*, $s_{\mathsf{init}} \in S_1$ is the *initial position*, $E \subseteq (S_1 \cup S_2) \times (S_1 \cup S_2)$ is the set of *edges* such that for all $s \in S_1 \cup S_2$, there exists $s' \in S_1 \cup S_2$ such that $(s, s') \in E$, $k \in \mathbb{N}$ is the *dimension* of the multi-weights, $w : E \to \mathbb{Z}^k$ is the *multi-weight labeling function*. G is a multi-weighted *one-player* game structure if $S_2 = \emptyset$.

A *play* in G is an infinite sequence of $\pi = s_0 s_1 \dots s_n \dots$ such that (i) $s_0 = s_{\mathsf{init}}$, (ii) for all $i \geq 0$ we have $(s_i, s_{i+1}) \in E$. A play $\pi = s_0 s_1 \dots s_n \dots$ is *ultimately periodic* if it can be decomposed as $\pi = \rho_1 \cdot \rho_2^\omega$ where ρ_1 and ρ_2 are two finite sequences of positions. The *prefix* up to position n of a play $\pi = s_0 s_1 \dots s_n \dots$ is the finite sequence $\pi(n) = s_0 s_1 \dots s_n$, its last element s_n is denoted by $\mathsf{Last}(\pi(n))$. A prefix $\pi(n)$ belongs to Player i $(i \in \{1, 2\})$ if $\mathsf{Last}(\pi(n)) \in S_i$. The set of plays in G is denoted by $\mathsf{Plays}(G)$, the corresponding set of prefixes is denoted by $\mathsf{Prefs}(G)$, the set of prefixes that belongs to Player i $(i \in \{1, 2\})$ is denoted by $\mathsf{Prefs}_i(G)$, and the set of ultimately periodic plays in G is denoted by $\mathsf{Plays}^{up}(G)$.

The *energy level vector* of a prefix of play $\rho = s_0 s_1 \dots s_n$ is $\mathsf{EL}(\rho) = \sum_{i=0}^{i=n-1} w(s_i, s_{i+1})$, and the *mean-payoff vector* of an ultimately periodic play $\pi = s_0 s_1 \dots s_n \dots$ is $\mathsf{MP}(\pi) = \lim_{n\to\infty} \frac{1}{n}\mathsf{EL}(\pi(n))$.

Strategies. A strategy for Player i $(i \in \{1, 2\})$ in G is a function $\lambda_i : \mathsf{Prefs}_i(G) \to S_1 \cup S_2$ such that for all $\rho \in \mathsf{Prefs}_i(G)$ we have $(\mathsf{Last}(\rho), \lambda_i(\rho)) \in E$. A play $\pi = s_0 s_1 \dots \in \mathsf{Plays}(G)$

is *consistent* with a strategy λ_i of Player i if $s_{j+1} = \lambda_i(s_0 s_1 \dots s_j)$ for all $j \geq 0$ such that $s_j \in S_i$. The *outcome of a pair of strategies*, λ_1 for Player 1 and λ_2 for Player 2, is the (unique) play which is consistent with both λ_1 and λ_2. We denote $\mathsf{outcome}_G(\lambda_1, \lambda_2)$ this outcome.

A strategy λ_1 for Player 1 has *finite-memory* if it can be encoded by a deterministic Moore machine $(M, m_0, \alpha_u, \alpha_n)$ where M is a finite set of states (the memory of the strategy), $m_0 \in M$ is the initial memory state, $\alpha_u : M \times (S_1 \cup S_2) \to M$ is an update function, and $\alpha_n : M \times S_i \to S_1 \cup S_2$ is the next-action function. If the game is in a Player-1 position $s \in S_1$ and $m \in M$ is the current memory value, then the strategy chooses $s' = \alpha_n(m, s)$ as the next position and the memory is updated to $\alpha_u(m, s)$. Formally, $\langle M, m_0, \alpha_u, \alpha_n \rangle$ defines the strategy λ such that $\lambda(\rho \cdot s) = \alpha_n(\hat{\alpha}_u(m_0, \rho), s)$ for all $\rho \in (S_1 \cup S_2)^*$ and $s \in S_1$, where $\hat{\alpha}_u$ extends α_u to sequences of positions as expected. A strategy is *memoryless* if $|M| = 1$. For a finite-memory strategy λ_1 of Player 1, let G_{λ_1} be the graph obtained as the product of G with the Moore machine defining λ_1, with initial vertex $\langle m_0, s_{\mathsf{init}} \rangle$ and where $(\langle m, s \rangle, \langle m', s' \rangle)$ is a transition in G_{λ_1} if $m' = \alpha_u(m, s)$, and either $s \in S_1$ and $s' = \alpha_n(m, s)$, or $s \in S_2$ and $(s, s') \in E$. The set of inifinite paths in G_{λ_1} and the set of plays consistent with λ_1 coincide. A similar definition can be given for the case of Player 2.

Objectives. An *objective* for Player 1 in G is a set of plays $W \subseteq \mathsf{Plays}(G)$. A strategy λ_1 for Player 1 is *winning* for W in G if for all plays in $\pi \in \mathsf{Plays}(G)$ that are consistent with λ_1, we have that $\pi \in W$. A strategy λ_2 for Player 2 is *spoiling* for W in G if for all plays in $\pi \in \mathsf{Plays}(G)$ that are consistent with λ_2, we have that $\pi \notin W$. We consider the following objectives:

- *Multi Energy objectives.* Given an initial energy vector $v_0 \in \mathbb{N}^k$, the *multi energy objective* $\mathsf{PosEnergy}_G(v_0) = \{\pi \in \mathsf{Plays}(G) \mid \forall n \geq 0 : v_0 + \mathsf{EL}(\pi(n)) \in \mathbb{N}^k\}$ requires that the energy level in all dimensions remains always nonnegative.
- *Multi Mean-payoff objectives.* Given a threshold vector $v \in \mathbb{Z}^k$, the *multi mean-payoff objective* $\mathsf{MeanPayoff}_G(v) = \{\pi \in \mathsf{Plays}^{up}(G) \mid \mathsf{MP}(\pi) \geq v\}$ requires for all dimensions j the mean-payoff for dimension j is at least $v(j)$.

Decision problems. We consider the following decision problems:

- The *unknown initial credit problem* asks, given an multi-weighted two-player game structure G, to decide whether there exists an initial credit vector $v_0 \in \mathbb{N}^k$ and a winning strategy λ_1 for Player 1 for the objective $\mathsf{PosEnergy}_G(v_0)$.
- The *mean-payoff threshold problem* (for finite memory) asks, given an multi-weighted two-player game structure G and a threshold vector $v \in \mathbb{Z}^k$, to decide whether there exists a *finite-memory* strategy λ_1 for Player 1 such that for all *finite-memory* strategies λ_2 of Player 2, $\mathsf{outcome}_G(\lambda_1, \lambda_2) \in \mathsf{MeanPayoff}_G(v)$.

Note that in the unknown initial credit problem, we allow arbitrary strategies (and we show in Theorem 5 that actually finite-memory strategies are sufficient), while in the mean-payoff threshold problem, we require finite-memory strategy which is restriction (according to Theorem 8) of a more general problem of deciding the existence of arbitrary winning strategies.

Determinacy and determinacy under finite-memory. A game G with an objective W is *determined* if either Player 1 has a winning strategy, or Player 2 has a spoiling strategy. A game G with an objective W is *determined under finite-memory* if either (a) Player 1 has a *finite-memory* strategy λ_1 such that for all *finite-memory* strategies λ_2 of Player 2, we have $\mathsf{outcome}_G(\lambda_1, \lambda_2) \in W$; or (b) Player 2 has a *finite-memory* strategy λ_2 such that for

all *finite-memory* strategies λ_1 of Player 1, we have $\mathsf{outcome}_G(\lambda_1, \lambda_2) \notin W$. Games with objectives W are determined (resp. determined under finite-memory) if all game structures with objectives W are determined (resp. determined under finite-memory). We say that determinacy and determined under finite-memory coincide for a class of objectives, if for all objectives in the class and all game structures, the answer of the determinacy and determined under finite-memory coincide (i.e., Player 1 has a winning strategy iff there is a finite-memory winning strategy, and similarly for Player 2). Generalized mean-payoff and energy objectives are measurable: (a) generalized mean-payoff objectives can be expressed as finite intersection of mean-payoff objectives and mean-payoff objectives are complete for the third level of Borel hierarchy [7]; and (b) generalized energy objectives can be expressed as finite intersection of energy objectives, and enery objectives are closed sets. Hence determinacy of generalized mean-payoff and energy games follows from the result of [15].

▶ **Theorem 2** (Determinacy [15]). *Generalized mean-payoff and energy games are determined.*

3 Determinacy under Finite-memory and Inter-reducibility

In this section, we establish four results. First, we show that to win generalized energy games, it is sufficient for Player 1 to play *finite-memory strategies*. Second, we show that to spoil generalized energy games, it is sufficient for Player 2 to play *memoryless strategies*. As a consequence, generalized energy games are determined under finite-memory. Third, using this finite-memory determinacy result, we show that the decision problems for generalized energy and mean-payoff games (see Section 2) are log-space inter-reducible. Finally, we show that infinite-memory strategies are more powerful than finite-memory strategies in generalized mean-payoff games.

For generalized energy games, we first show that finite-memory strategies are sufficient for Player 1, and then that memoryless strategies are sufficient for Player 2.

▶ **Lemma 3.** *For all multi-weighted two-player game structures G, the answer to the unknown initial credit problem is* YES *iff there exists a initial credit $v_0 \in \mathbb{N}^k$ and a finite-memory strategy λ_1^{FM} for Player 1 such that for all strategies λ_2 of Player 2, $\mathsf{outcome}_G(\lambda_1^{\mathsf{FM}}, \lambda_2) \in \mathsf{PosEnergy}_G(v_0)$.*

Proof. One direction is trivial. For the other direction, assume that λ_1 is a (not necessary finite-memory) winning strategy for Player 1 in G with initial credit $v_0 \in \mathbb{N}^k$. We show how to construct from λ_1 a finite-memory strategy λ_1^{FM} which is winning against all strategies of Player 2 for initial credit v_0. For that we consider the unfolding of the game graph G in which Player 1 plays according to λ_1. This infinite tree, noted $T_{G(\lambda_1)}$, has as set of nodes all the prefixes of plays in G when Player 1 plays according to λ_1. We associate to each node $\rho = s_0 s_1 \dots s_n$ in this tree the energy vector $v_0 + \mathsf{EL}(\rho)$. As λ_1 is winning, we have that $v_0 + \mathsf{EL}(\rho) \in \mathbb{N}^k$ for all ρ. Now, consider the set $(S_1 \cup S_2) \times \mathbb{N}^k$, and the relation $\sqsubseteq$ on this set defined as follows: $(s_1, v_1) \sqsubseteq (s_2, v_2)$ iff $s_1 = s_2$ and $v_1 \leq v_2$ i.e., for all i, $1 \leq i \leq k$, $v_1(i) \leq v_2(i)$. The relation $\sqsubseteq$ is a wqo (easy consequence of Lemma 1). As a consequence, on every infinite branch $\pi = s_0 s_1 \dots s_n \dots$ of $T_{G(\lambda_1)}$, there exists two positions $i < j$ such that $\mathsf{Last}(\pi(i)) = \mathsf{Last}(\pi(j))$ and $\mathsf{EL}(\pi(i)) \leq \mathsf{EL}(\pi(j))$. We say that node j subsumes node i. Now, let $T^{\mathsf{FM}}_{G(\lambda_1)}$ be the tree $T_{G(\lambda_1)}$ where we stop each branch when we reach a node n_2 which subsumes one of its ancestor node n_1. Clearly, $T^{\mathsf{FM}}_{G(\lambda_1)}$ is finite. Also, it is easy to see that Player 1 can play in the subtree rooted at n_2 as she plays in the subtree rooted in n_1 because its energy level in n_2 is greater than in n_1. From $T^{\mathsf{FM}}_{G(\lambda_1)}$, we can construct a Moore

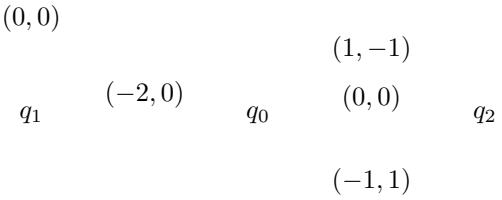


Figure 1 Player 1 (round states) wins with initial credit $(2, 0)$ when Player 2 (square states) can use memoryless strategies, but not when Player 2 can use arbitrary strategies.

machine which encode a finite-memory strategy λ_1^{FM} which is winning the generalized energy game G as it is winning for initial energy level v_0. □

▶ **Lemma 4.** *[4] For all multi-weigthed two-player game structures G, the answer to the unknown initial credit problem is* No *if and only if there exists a* memoryless *strategy λ_2 for Player 2, such that for all initial credit vectors $v_0 \in \mathbb{N}^k$ and all strategies λ_1 for Player 1 we have* $\mathsf{outcome}_G(\lambda_1, \lambda_2) \notin \mathsf{PosEnergy}_G(v_0)$.

As a consequence of the two previous lemmas, we have the following theorem.

▶ **Theorem 5.** *Generalized energy games are determined under finite-memory, and determinacy coincide with determinacy under finite-memory for generalized energy games.*

▶ Remark. Note that even if Player 2 can be restricted to play memoryless strategies in generalized energy games, it may be that Player 1 is winning with some initial credit vector v_0 when Player 2 is memoryless, and is not winning with the same initial credit vector v_0 when Player 2 can use arbitrary strategies. This situation is illustrated in Figure 1 where Player 1 (owning round states) can maintain the energy nonegative in all dimensions with initial credit $(2, 0)$ when Player 2 (owning square states) is memoryless. Indeed, either Player 2 chooses the left edge from q_0 to q_1 and Player 1 wins, or Player 2 chooses the right edge from q_0 to q_2, and Player 1 wins as well by alternating the edges back to q_0. Now, if Player 2 has memory, then Player 2 wins by choosing first the right edge to q_2, which forces Player 1 to come back to q_0 with multi-weight $(-1, 1)$. The energy level is now $(1, 1)$ in q_0 and Player 2 chooses the left edge to q_1 which is losing for Player 1. Note that Player 1 wins with initial credit $(2, 1)$ and $(3, 0)$ (or any larger credit) against all arbitrary strategies of Player 2.

We now show that generalized mean-payoff games (where players are restricted to play finite-memory strategies by definition) are log-space equivalent to generalized energy games. First note that the mean-payoff threshold problem with threshold vector $v \in \mathbb{Z}^k$ can be reduced to the mean-payoff threshold problem with threshold vector $\{0\}^k$, by shifting all multi-weights in the game graph by v (which has the effect of shifting the mean-payoff value by v). Given this reduction, the following result shows that the unknown initial credit problem (for multi-energy games) and the mean-payoff threshold problem (with finite-memory strategies) are equivalent.

▶ **Theorem 6.** *For all multi-weigthed two-player game structures G with dimension k, the answer to the unknown initial credit problem is* Yes *if and only if the answer to the mean-payoff threshold problem (for finite memory) with threshold vector $\{0\}^k$ is* Yes.

Proof. First, assume that there exists a winning strategy λ_1 for Player 1 in G for the multi energy objective $\mathsf{PosEnergy}_G(v_0)$ (for some v_0). Theorem 5 establishes that finite memory is

sufficient to win multi-energy games, so we can assume that λ_1 has finite memory. Consider the restriction of the graph G_{λ_1} to the reachable vertices, and we show that the energy vector of every simple cycle is nonnegative. By contradcition, if there exists a simple cycle with energy vector negative in one dimension, then the infinite path that reaches this cycle and loops through it forever would violate the objective $\mathsf{PosEnergy}_G(v_0)$ regardless of the vector v_0.

Now, this shows that every reachable cycle in G_{λ_1} has nonnegative mean-payoff value in all dimensions, hence λ_1 is winning for the multi mean-payoff objective $\mathsf{MeanPayoff}_G(\{0\}^k)$.

Second, assume that there exists a finite-memory strategy λ_1 for Player 1 that is winning in G for the multi mean-payoff objective $\mathsf{MeanPayoff}_G(\{0\}^k)$. By the same argument as above, all simple cycles in G_{λ_1} are nonnegative and the strategy λ_1 is also winning for the objective $\mathsf{PosEnergy}_G(v_0)$ for some v_0. Taking $v_0 = \{nW\}^k$ where n is the number of states in G_{λ_1} (which bounds the length of the acyclic paths) and $W \in \mathbb{Z}$ is the largest weight in the game suffices. □

Note that the result of Theorem 6 does not hold for arbitrary strategies as shown in the following lemma.

▶ **Lemma 7.** *In generalized mean-payoff games, infinite memory may be necessary to win (finite-memory strategies may not be sufficient).*

Proof. To show this, we first need to define the mean-payoff vector of arbitrary plays (because arbitrary strategies, i.e., infinite-memory strategies, may produce non-ultimately periodic plays). In particular, the limit of $\frac{1}{n} \cdot \mathsf{EL}(\pi(n))$ for $n \to \infty$ may not exist for arbitrary plays π. Therefore, two possible definitions are usually considered, namely either $\underline{\mathsf{MP}}(\pi) = \liminf_{n\to\infty} \frac{1}{n} \cdot \mathsf{EL}(\pi(n))$, or $\overline{\mathsf{MP}}(\pi) = \limsup_{n\to\infty} \frac{1}{n} \cdot \mathsf{EL}(\pi(n))$. In both cases, better payoff can be obtained with infinite memory: the example of Figure 2 shows a game where all states belong to Player 1. We claim that (a) for $\underline{\mathsf{MP}}$, Player 1 can achieve a threshold vector $(1, 1)$, and (b) for $\overline{\mathsf{MP}}$, Player 1 can achieve a threshold vector $(2, 2)$; (c) if we restrict Player 1 to use a finite-memory strategy, then it is not possible to win the multi mean-payoff objective with threshold $(1, 1)$ (and thus also not with $(2, 2)$). To prove (a), consider the strategy that visits n times q_a and then n times q_b, and repeats this forever with increasing value of n. This guarantees a mean-payoff vector $(1, 1)$ for $\underline{\mathsf{MP}}$ because in the long-run roughly half of the time is spent in q_a and roughly half of the time in q_b. To prove (b), consider the strategy that alternates visits to q_a and q_b such that after the nth alternation, the self-loop on the visited state q ($q \in \{q_a, q_b\}$) is taken so many times that the average frequency of q gets larger than $\frac{1}{n}$ in the current finite prefix of the play. This is always possible and achieves threshold $(2, 2)$ for $\overline{\mathsf{MP}}$. Note that the above two strategies require infinite memory. To prove (c), notice that finite-memory strategies produce an ultimately periodic play and therefore $\underline{\mathsf{MP}}$ and $\overline{\mathsf{MP}}$ coincide with MP. It is easy to see that such a play cannot achieve $(1, 1)$ because the periodic part would have to visit both q_a and q_b and then the mean-payoff vector (v_1, v_2) of the play would be such that $v_1 + v_2 < 2$ and thus $v_1 = v_2 = 1$ is impossible. □

Theorem 6 and Lemma 7, along with Theorem 5 gives the following result.

▶ **Theorem 8.** *Generalized mean-payoff games are determined under finite-memory, however determinacy and determined under finite-memory do not coincide for generalized mean-payoff games.*

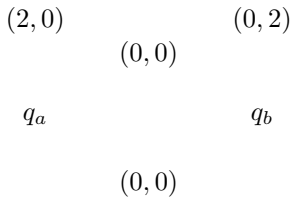


Figure 2 A generalized mean-payoff game where infinite memory is necessary to win (Lemma 7).

4 coNP-completeness for Finite-Memory Strategies

In this section, we present a nondeterministic polynomial time algorithm to recognize the instances for which there is no winning strategies for Player 1 in a multi-energy game. First, we show that the one-player version of this game can be solved by checking the existence of a circuit (i.e., a not necessarily simple cycle) with overall nonnegative effect in all dimensions. Second, we build on this and the memoryless result for Player 2 to define a coNP algorithm. The main result (Theorem 9) is derived from Lemma 11 and Lemma 12 below.

▶ **Theorem 9.** *The unknown initial credit and the mean-payoff threshold problems for multi-weighted two-player game structures are coNP-complete.*

coNP upper bound. First, we need the following result about finding zero circuits in multi-weighted directed graphs (a graph is a one-player game). A zero circuit is a finite sequence $s_0 s_1 \dots s_n$ such that $s_0 = s_n$, $(s_i, s_{i+1}) \in E$ for all $0 \leq i < n$, and $\sum_{i=0}^{n-1} w(s_i, s_{i+1}) = (0, 0, \dots, 0)$. The circuit need not be simple.

▶ **Lemma 10** ([14])**.** *Determining if a k-dimensional directed graph contains a zero circuit can be done in polynomial time.*

▶ **Lemma 11.** *The unknown initial credit and the mean-payoff threshold problems for multi-weighted two-player game structures are in coNP.*

Proof. By Lemma 4, we know that Player 2 can be restricted to play memoryless strategies. A coNP algorithm can guess a memoryless strategy λ and check in polynomial time that it is winning using the following argument.

First, consider the graph G_λ as a one-player game (in which all states belong to player 1. We show that if there exists an initial energy level v_0 and an infinite play $\pi = s_0 s_1 \dots s_n \dots$ in G_λ such that $\pi \in \mathsf{PosEnergy}(v_0)$ then there exist a reachable circuit in G_λ that has nonnegative effect in all dimensions. To show that, we extend π with the energy information as follows: $\pi' = (s_0, w_0)(s_1, w_1) \dots (s_n, w_n) \dots$ where $w_0 = v_0$ and for all $i \geq 1$, $w_i = v_0 + \mathsf{EL}(\pi(i))$. As $\pi \in \mathsf{PosEnergy}(v_0)$, we know that for all $i \geq 0$, $w_i \in \mathbb{N}^k$. So, we can define the following order on the pairs $(s, w) \in (S_1 \cup S_2) \times \mathbb{N}^k$ in the run: $(s, w) \sqsubseteq (s', w')$ iff $s = s'$ and $w(j) \leq w'(j)$ for all $1 \leq j \leq k$. From Lemma 1, it is easy to show that $\sqsubseteq$ is a wqo. Then there exist two positions $i_1 < i_2$ in π' such that $(s_{i_1}, w_{i_1}) \sqsubseteq (s_{i_2}, w_{i_2})$. The circuit underlying those two positions has nonnegative effect in all dimensions.

Based on this, we can decide if there exists an initial energy vector v_0 and an infinite path in G_λ that satisfies $\mathsf{PosEnergy}_G(v_0)$ using the result of Lemma 10 on modified version of G_λ obtained as follows. In every state of G_λ, we add k self-loops with respective multi-weight $(-1, 0, \dots, 0)$, $(0, -1, 0, \dots, 0)$, $\dots$, $(0, \dots, 0, -1)$, i.e. each self-loop removes one unit of energy in one dimension. It is easy to see that G_λ has a circuit with nonnegative effect in all

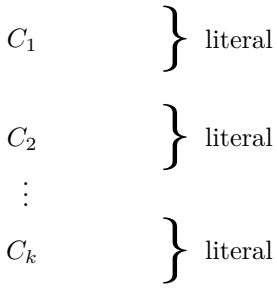


Figure 3 Game graph construction for a 3SAT formula (Lemma 12).

dimensions if and only if the modified G_λ has a zero circuit, which can be determined in polynomial time. The result follows. □

Lower bound: coNP-hardness. We show that the unknown initial credit problem for multi-weighted two-player game structures is coNP-hard. We present a reduction from the complement of the 3SAT problem which is NP-complete [16].

Hardness proof. We show that the problem of deciding whether Player 1 has a winning strategy for the unknown initial credit problem for multi-weighted two-player game structures is at least as hard as deciding whether a 3SAT formula is unsatisfiable. Consider a 3SAT formula ψ in CNF with clauses $C_1, C_2, \ldots, C_k$ over variables $\{x_1, x_2, \ldots, x_n\}$, where each clause consists of disjunctions of exactly three literals (a literal is a variable or its complement). Given the formula ψ, we construct a game graph as shown in Figure 3. The game graph is as follows: from the initial position, Player 1 chooses a clause, then from a clause Player 2 chooses a literal that appears in the clause (i.e., makes the clause true). From every literal the next position is the initial position. We now describe the multi-weight labeling function w. In the multi-weight function there is a component for every literal. For edges from the initial position to the clause positions, and from the clause positions to the literals, the weight for every component is 0. We now define the weight function for the edges from literals back to the initial position: for a literal y, and the edge from y to the initial position, the weight for the component of y is 1, the weight for the component of the complement of y is -1, and for all the other components the weight is 0. We now define a few notations related to assignments of truth values to literals. We consider *assignments* that assign truth values to all the literals. An assignment is *valid* if for every literal the truth value assigned to the literal and its complement are complementary (i.e., for all $1 \leq i \leq n$, if x_i is assigned true (resp. false), then the complement $\overline{x}_i$ of x_i is assigned false (resp. true)). An assignment that is not valid is *conflicting* (i.e., for some $1 \leq i \leq n$, both x_i and $\overline{x}_i$ are assigned the same truth value). If the formula ψ is satisfiable, then there is a valid assignment that satisfies all the clauses. If the formula ψ is not satisfiable, then every assignment that satisfies all the clauses must be conflicting. We now present two directions of the hardness proof.

ψ satisfiable implies Player 2 winning. We show that if ψ is satisfiable, then Player 2 has a memoryless winning strategy. Since ψ is satisfiable, there is a valid assignment A that satisfies every clause. The memoryless strategy is constructed from the assignment A as follows: for a clause C_i, the strategy chooses a literal as successor that appears in C_i and is set to true by the assignment. Consider an arbitrary strategy for Player 1, and the infinite play: the literals visited in the play are all assigned truth values true by A, and the infinite play must visit some literal infinitely often. Consider the literal x that appears infinitely

often in the play, then the complement literal $\overline{x}$ is never visited, and every time literal x is visited, the component corresponding to $\overline{x}$ decreases by 1, and since x appears infinitely often it follows that the play is winning for Player 2 for every finite initial credit. It follows that the strategy for Player 2 is winning, and the answer to the unknown initial credit problem is "No".

ψ not satisfiable implies Player 1 is winning. We now show that if ψ is not satisfiable, then Player 1 is winning. By determinacy, it suffices to show that Player 2 is not winning, and by existence of memoryless winning strategy for Player 2 (Lemma 4), it suffices to show that there is no memoryless winning strategy for Player 2. Fix an arbitrary memoryless strategy for Player 2, (i.e., in every clause Player 2 chooses a literal that appears in the clause). If we consider the assignment A obtained from the memoryless strategy, then since ψ is not satisfiable it follows that the assignment A is conflicting. Hence there must exist clause C_i and C_j and variable x_k such that the strategy chooses the literal x_k in C_i and the complement variable $\overline{x}_k$ in C_j. The strategy for Player 1 that at the starting position alternates between clause C_i and C_j, along with that the initial credit of 1 for the component of x_k and $\overline{x}_k$, and 0 for all other components, ensures that the strategy for Player 2 is not winning. Hence the answer to the unknown initial credit problem is "Yes", and we have the following result.

▶ **Lemma 12.** *The unknown initial credit and the mean-payoff threshold problems for multi-weighted two-player game structures are coNP-hard.*

Observe that our hardness proof works with weights restricted to the set $\{-1, 0, 1\}$.

5 NP-completeness for Memoryless Strategies

In this section we consider the unknown initial credit and the mean-payoff threshold problems for multi-weighted two-player game structures when Player 1 is restricted to use memoryless strategies. We will show NP-completeness for these problems.

▶ **Lemma 13.** *The unknown intial credit and the mean-payoff threshold problems for multi-weighted two-player game structures for memoryless strategies for Player 1 lie in NP.*

Proof. The inclusion in NP is obtained as follows: the polynomial witness is the memoryless strategy for Player 1, and once the strategy is fixed we obtain a game graph with choices for Player 2 only. The verification problem for the unknown initial credit checks that for every dimension there is no negative cycle, and the verification problem for mean-payoff threshold checks that for every dimension every cycle satisfy the threshold condition. Both the above verification problem can be achieved in polynomial time by solving the energy-game and mean-payoff game problem on graphs with choices for Player 2 only [13, 3, 6]. The desired result follows. □

Lemma 14 shows NP-hardness for dimension $k = 2$ and arbitrary integral weights, and is obtained by a reduction from the KNAPSACK problem. If $k = 1$, then the problems reduces to the classical energy and mean-payoff games, and is in NP ∩ coNP [3, 6, 20] (so the hardness result cannot be obtained for $k = 1$).

▶ **Lemma 14.** *The unknown intial credit and the mean-payoff threshold problems for multi-weighted two-player game structures for memoryless strategies for Player 1 are NP-hard, even in one-player game structures with dimension $k = 2$ for the weight function.*

In Lemma 15 we show the hardness of the problem when the weights are in $\{-1, 0, 1\}$, but the dimension is arbitrary. It has been shown in [12] that if the weights are $\{-1, 0, 1\}$ and the dimension is 2, then the problem can be solved in polynomial time.

▶ **Lemma 15.** *The unknown intial credit and the mean-payoff threshold problems for multi-weighted two-player game structures for memoryless strategies for Player 1 are NP-hard, even in one-player game structures when weights are restricted to* $\{-1, 0, 1\}$.

Proof. We present a reduction from the 3SAT problem. Consider a 3SAT formula Φ over a set $X = \{x_1, x_2, \ldots, x_n\}$ of variables, and a set $C_1, C_2, \ldots, C_m$ of clauses such that each clause has 3-literals (a literal is a variable or its complement). We construct a one-player game structure with a weight function of dimension m from Φ. The set of positions is $S_1 = X \cup \{(x_i, j) \mid x_i \in X, j \in \{T, F\}\} \cup \{x_{n+1}\}$ and $S_2 = \emptyset$. The set of edges is as follows: $E = \{(x_i, (x_i, T)), (x_i, (x_i, F)) \mid x_i \in X\} \cup \{((x_i, T), x_{i+1}), ((x_i, F), x_{i+1}) \mid x_i \in X\} \cup \{(x_{n+1}, x_1)\}$. Intuitively, in the game structure, for every variable Player 1 has a choice to set x_i as "True" (edge from x_i to (x_i, T)), and choice to set x_i as "False" (edge from x_i to (x_i, F)). From (x_i, T) and (x_i, F) the next position is x_{i+1}, and from the position x_{n+1} the next position is x_1. The weight function $w : E \to \mathbb{Z}^m$ has m dimensions: (a) for an edge $e = (x_i, (x_i, T))$ (resp. $e = (x_i, (x_i, F))$) and $1 \leq k \leq m$, the k-th component of $w(e)$ is 1 if the choice x_i as "True" (resp. "False") satisfies clause C_k, and otherwise the k-th component is 0; (b) for edges $e = ((x_i, j), x_{i+1})$, with $j \in \{T, F\}$, every component of $w(e)$ is 0; and (c) for the edge $e = (x_{n+1}, x_1)$, for all $1 \leq k \leq m$, the k-th component of $w(e) = -1$. If Φ is satisfiable, then consider a satisfying assignment A, and we construct a memoryless strategy λ_1 as follows: for a position x_i, if $A(x_i)$ is "True", then choose (x_i, T), otherwise choose (x_i, F). The memoryless strategy λ_1 with initial credit vector $\{0\}^m$ ensures that the answer to the unknown initial credit problem for memoryless strategies is "Yes". Conversely, if there is a memoryless strategy λ_1 for the unknown initial credit problem, then the memoryless strategy must satisfy every clause. A satisfying assignment A for Φ is as follows: $A(x_i)$ is "True" if $\lambda_1(x_i) = (x_i, T)$, and "False", otherwise. It follows that Φ is satisfiable iff the answer to the unknown initial credit problem for memoryless strategies is "Yes". The argument for the mean-payoff threshold problem is analogous. The desired result follows. □

The following theorem follows from the results of Lemma 13, Lemma 14 and Lemma 15.

▶ **Theorem 16.** *The unknown initial credit and the mean-payoff threshold problems for multi-weighted two-player game structures for memoryless strategies for Player 1 are NP-complete.*

6 Conclusion

In this work we considered games with multiple mean-payoff and energy objectives, and established determinacy under finite-memory, inter-reducibility of these two classes of games for finite-memory strategies, and improved the complexity bounds from EXPSPACE to coNP-complete.

Two interesting problems are open: (A) for generalized mean-payoff games, the winning strategies with infinite memory are more powerful than finite-memory strategies, and the complexity of solving generalized mean-payoff games with infinite-memory strategies remains open. (B) it is not knwon how to compute the exact or approximate Pareto curve (trade-off curve) for multi-objective mean-payoff and energy games.

Acknowledgement. We are grateful to Jean Cardinal for pointing the reference [14].

References

1 M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *Proc. of ICALP*, LNCS 372, pages 1–17. Springer, 1989.

2 R. Alur, A. Degorre, O. Maler, and G. Weiss. On omega-languages defined by mean-payoff conditions. In *Proc. of FOSSACS*, LNCS 5504, pages 333–347. Springer, 2009.

3 P. Bouyer, U. Fahrenberg, K. G. Larsen, N. Markey, and J. Srba. Infinite runs in weighted timed automata with energy constraints. In *Proc. of FORMATS*, LNCS 5215, pages 33–47. Springer, 2008.

4 T. Brázdil, P. Jancar, and A. Kucera. Reachability games on extended vector addition systems with states. In *Proc. of ICALP*, LNCS 6199, pages 478–489. Springer, 2010.

5 L. Brim, J. Chaloupka, L. Doyen, R. Gentilini, and J.-F. Raskin. Faster algorithms for mean-payoff games (submitted for publication). Technical report, 2010.

6 A. Chakrabarti, L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Resource interfaces. In *Proc. of EMSOFT: Embedded Software*, LNCS 2855, pages 117–133. Springer, 2003.

7 K. Chatterjee. Concurrent games with tail objectives. *Theor. Comput. Sci.*, 388(1-3):181–198, 2007.

8 K. Chatterjee, L. Doyen, H. Edelsbrunner, T. A. Henzinger, and P. Rannou. Mean-payoff automaton expressions. In *Proc. of CONCUR*, LNCS 6269, pages 269–283. Springer, 2010.

9 K. Chatterjee, L. Doyen, T. Henzinger, and J.-F. Raskin. Generalized mean-payoff and energy games. *CoRR*, 2010. http://arxiv.org/abs/1007.1669.

10 A. Condon. The complexity of stochastic games. *Inf. Comput.*, 96(2):203–224, 1992.

11 A. Ehrenfeucht and J. Mycielski. Positional strategies for mean-payoff games. *Int. J. of Game Theory*, 8:109–113, 1979.

12 Chaloupka J. Z-reachability problem for games on 2-dimensional vector addition systems with states is in P. In *Proceedings of RP 2010: Reachability Problems*, LNCS 6227, pages 104–119. Springer-Verlag, 2010.

13 R.M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

14 S. R. Kosaraju and G. F. Sullivan. Detecting cycles in dynamic graphs in polynomial time (preliminary version). In *Proc. of STOC: Symposium on Theory of Computing*, pages 398–406. ACM, 1988.

15 D. Martin. Borel determinacy. In *Annals of Mathematics*, volume 102, pages 363–371, 1975.

16 C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1993.

17 A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. of POPL*, pages 179–190, 1989.

18 P. J. Ramadge and W. M. Wonham. Supervisory control of a class of discrete event processes. *SIAM Journal on Control and Optimization*, 25(1):206–230, 1987.

19 L. S. Shapley. Stochastic games. In *Proc. of the National Acadamy of Science USA*, volume 39, pages 1095–1100, 1953.

20 U. Zwick and M. Paterson. The complexity of mean payoff games on graphs. *Th. Comp. Sc.*, 158:343–359, 1996.